

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

January 2, 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`).

By default, the package `nicematrix` doesn't patch any existing code.

However, when the option `renew-dots` is used, the commands `\cdots`, `\ldots`, `\dots`, `\vdots`, `\ddots` and `\iddots` are redefined in the environments provided by `nicematrix`. In the same way, if the option `renew-matrix` is used, the environment `{matrix}` of `amsmath` is redefined.

On the other hand, the environment `{array}` is never redefined.

Of course, the package `nicematrix` uses the features of the package `array`. It tries to be independent of its implementation. Unfortunately, it was not possible to be strictly independent. For example, the package `nicematrix` relies upon the fact that the package `{array}` uses `\ialign` to begin the `\halign`.

## 1 Declaration of the package and packages loaded

The prefix `nicematrix` has been registred 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}
```

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

We load some packages.

```
9 \RequirePackage { array }
10 \RequirePackage { amsmath }
```

---

\*This document corresponds to the version 6.26c of `nicematrix`, at the date of 2024/01/02.

```

11 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
12 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
13 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
14 \cs_generate_variant:Nn \@@_error:nn { n e }
15 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
16 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
17 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
18 \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).

```

19 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
20 {
21   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
22     { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
23     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
24 }

```

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

```

25 \cs_new_protected:Npn \@@_error_or_warning:n
26 { \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”.

```

27 \bool_new:N \g_@@_messages_for_Overleaf_bool
28 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
29 {
30   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
31   || \str_if_eq_p:on \c_sys_jobname_str { output } % for Overleaf
32 }

```

```

33 \cs_new_protected:Npn \@@_msg_redirect_name:nn
34 { \msg_redirect_name:nnn { nicematrix } }
35 \cs_new_protected:Npn \@@_gredirect_none:n #1
36 {
37   \group_begin:
38   \globaldefs = 1
39   \@@_msg_redirect_name:nn { #1 } { none }
40   \group_end:
41 }
42 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
43 {
44   \@@_error:n { #1 }
45   \@@_gredirect_none:n { #1 }
46 }
47 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
48 {
49   \@@_warning:n { #1 }
50   \@@_gredirect_none:n { #1 }
51 }

```

## 2 Security test

Within the package `nicematrix`, we will have to test whether a cell of a `{NiceTabular}` is empty. For the cells of the columns of type p, b, m, X and V, we will test whether the cell is syntactically empty

(that is to say that there is only spaces between the ampersands &). That test will be done with the command `\@@_test_if_empty:` by testing if the two first tokens in the cells are (during the TeX process) are `\ignorespaces` and `\unskip`.

However, if, one day, there is a changement in the implementation of `array`, maybe that this test will be broken (and `nicematrix` also).

That's why, by security, we will take a test in a small `{tabular}` composed in the box `\l_tmpa_box` used as sandbox.

```

52 \@@_msg_new:nn { Internal~error }
53 {
54   Potential~problem~when~using~nicematrix.\\
55   The~package~nicematrix~have~detected~a~modification~of~the~
56   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
57   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
58   this~message~again,~load~nicematrix~with:~\token_to_str:N
59   \usepackage[no-test-for-array]{nicematrix}.
60 }

61 \@@_msg_new:nn { mdwtab~loaded }
62 {
63   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
64   This~error~is~fatal.
65 }

66 \cs_new_protected:Npn \@@_security_test:n #1
67 {
68   \peek_meaning:NTF \ignorespaces
69   { \@@_security_test_i:w }
70   { \@@_error:n { Internal~error } }
71   #1
72 }

73 \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
74 {
75   \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
76   #1
77 }

```

Here, the box `\l_tmpa_box` will be used as sandbox to take our security test. This code has been modified in version 6.18 (see question 682891 on TeX StackExchange).

```

78 \hook_gput_code:nnn { begindocument / after } { . }
79 {
80   \IfPackageLoadedTF { mdwtab }
81   { \@@_fatal:n { mdwtab~loaded } }
82   {
83     \bool_if:NF \g_@@_no_test_for_array_bool
84     {
85       \group_begin:
86       \hbox_set:Nn \l_tmpa_box
87       {
88         \begin { tabular } { c > { \@@_security_test:n } c c }
89         text & & text
90         \end { tabular }
91       }
92       \group_end:
93     }
94   }
95 }

```

### 3 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`).

```

96 \cs_new_protected:Npn \@@_collect_options:n #1
97 {
98   \peek_meaning:NTF [
99     { \@@_collect_options:nw { #1 } }
100     { #1 { } }
101   }

```

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

```

102 \NewDocumentCommand \@@_collect_options:nw { m r[] }
103 { \@@_collect_options:nn { #1 } { #2 } }
104
105 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
106 {
107   \peek_meaning:NTF [
108     { \@@_collect_options:nnw { #1 } { #2 } }
109     { #1 { #2 } }
110   }
111
112 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
113 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

### 4 Technical definitions

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

```

114 \tl_const:Nn \c_@@_b_tl { b }
115 \tl_const:Nn \c_@@_c_tl { c }
116 \tl_const:Nn \c_@@_l_tl { l }
117 \tl_const:Nn \c_@@_r_tl { r }
118 \tl_const:Nn \c_@@_all_tl { all }
119 \tl_const:Nn \c_@@_dot_tl { . }
120 \tl_const:Nn \c_@@_default_tl { default }
121 \tl_const:Nn \c_@@_star_tl { * }
122 \str_const:Nn \c_@@_r_str { r }
123 \str_const:Nn \c_@@_c_str { c }
124 \str_const:Nn \c_@@_l_str { l }
125 \str_const:Nn \c_@@_R_str { R }
126 \str_const:Nn \c_@@_C_str { C }
127 \str_const:Nn \c_@@_L_str { L }
128 \str_const:Nn \c_@@_j_str { j }
129 \str_const:Nn \c_@@_si_str { si }

```

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).

```

130 \tl_new:N \l_@@_argspec_tl
131 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
132 \cs_generate_variant:Nn \str_lowercase:n { V }

133 \hook_gput_code:nnn { begindocument } { . }
134 {
135   \IfPackageLoadedTF { tikz }
136   {

```

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

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

```

137   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
138   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
139 }
140 {
141   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
142   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
143 }
144 }

```

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 May 2023, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

145 \IfClassLoadedTF { revtex4-1 }
146 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
147 {
148   \IfClassLoadedTF { revtex4-2 }
149   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
150   {

```

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

```

151   \cs_if_exist:NT \rvtx@ifformat@geq
152   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
153   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
154 }
155 }

```

```

156 \cs_generate_variant:Nn \tl_if_single_token_p:n { V }

```

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.

```

157 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
158 {
159   \iow_now:Nn \@mainaux
160   {
161     \ExplSyntaxOn
162     \cs_if_free:NT \pgfsyspdfmark
163     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
164     \ExplSyntaxOff

```

```

165     }
166     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
167 }

```

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.

```

168 \ProvideDocumentCommand \iddots { }
169 {
170   \mathinner
171   {
172     \tex_mkern:D 1 mu
173     \box_move_up:nn { 1 pt } { \hbox { . } }
174     \tex_mkern:D 2 mu
175     \box_move_up:nn { 4 pt } { \hbox { . } }
176     \tex_mkern:D 2 mu
177     \box_move_up:nn { 7 pt }
178     { \vbox:n { \kern 7 pt \hbox { . } } }
179     \tex_mkern:D 1 mu
180   }
181 }

```

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.

```

182 \hook_gput_code:nnn { begindocument } { . }
183 {
184   \IfPackageLoadedTF { booktabs }
185   { \iow_now:Nn \mainaux \nicematrix@redefine@check@rerun }
186   { }
187 }
188 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
189 {
190   \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`).

```

191   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
192   {
193     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
194     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
195   }
196 }

```

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

```

197 \hook_gput_code:nnn { begindocument } { . }
198 {
199   \IfPackageLoadedTF { colortbl }
200   { }
201   {

```

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

```

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

```

Idem for `\CT@drs@`.

```

209     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
210     \cs_set:Npn \CT@drs #1 #2
211     {
212         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
213         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
214     }
215     \cs_set:Npn \hline
216     {
217         \noalign { \ifnum 0 = ` } \fi
218         \cs_set_eq:NN \hskip \vskip
219         \cs_set_eq:NN \vrule \hrule
220         \cs_set_eq:NN \@width \@height
221         { \CT@arc@ \vline }
222         \futurelet \reserved@a
223         \@xhline
224     }
225 }
226 }
```

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

```

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

```

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

```

237     \skip_horizontal:N \c_zero_dim
238 }
```

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

```

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

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

```

243 \cs_set:Npn \@@_cline
```

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

```

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

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

```

245 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
246 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
247 {
```

---

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

```

248 \tl_if_empty:nTF { #3 }
249 { \@@_cline_iii:w #1|#2-#2 \q_stop }
250 { \@@_cline_ii:w #1|#2-#3 \q_stop }
251 }
252 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
253 { \@@_cline_iii:w #1|#2-#3 \q_stop }
254 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
255 {

```

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).

```

256 \int_compare:nNt { #1 } < { #2 }
257 { \multispan { \int_eval:n { #2 - #1 } } & }
258 \multispan { \int_eval:n { #3 - #2 + 1 } }
259 {
260 \CT@arc@
261 \leaders \hrule \@height \arrayrulewidth \hfill
262 \skip_horizontal:N \c_zero_dim
263 }

```

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

```

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

```

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

```

269 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

```

```

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

```

```

280 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
281 {
282 \tl_if_head_eq_meaning:nNTF { #1 } [
283 { \cs_set:Npn \CT@drsc@ { \color #1 } }
284 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
285 ]
286 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }

```

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

```

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

```

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

```

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

297 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor

298 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
299   {
300     \tl_set_rescan:Nno
301       #1
302     {
303       \char_set_catcode_other:N >
304       \char_set_catcode_other:N <
305     }
306     #1
307   }

```

Since we will do ourself the expansion of the preamble of the array, we will modify `\@mkpream` of array in order to skip the operation of expansion done by `\@mkpream`.

```

308 \cs_set_eq:NN \@@_old_mkpream: \@mkpream
309 \cs_set_protected:Npn \@@_mkpream: #1
310   {

```

The command `\@@_mkpream_colortbl:` will be empty when `colortbl` is not loaded.

```

311   \@mkpream_colortbl:
312   \gdef\@preamble{} \lastchclass 4 \@firstampttrue
313   \let\@sharp\relax
314   \def\@startpbox##1{\unexpanded\expandafter{\expandafter
315     \@startpbox\expandafter{##1}}}\let\@endpbox\relax
316   \let\do@row@strut\relax
317   \let\ar@align@mcell\relax
318   \@temptokena{#1} % \@tempswatrue
319   % \@whilesw@if@tempswa\fi{\@tempswafalse\the\NC@list}%
320   \count@m@ne
321   \let\the@toks\relax
322   \prepnext@tok

```

We have slightly modified the code of the original version of `\@mkpream` in order to have something compatible with `\ExplSyntaxOn`.

```

323   \exp_args:NV \tl_map_variable:NNn \@temptokena \@nextchar
324   {\@testpach
325     \ifcase \@chclass \@classz \or \@classi \or \@classii
326       \or \save@decl \or \or \@classv \or \@classvi
327       \or \@classvii \or \@classviii
328       \or \@classx
329       \or \@classx \fi
330     \@lastchclass\@chclass}%
331   \ifcase\@lastchclass
332     \@acol \or
333     \or
334     \@acol \or
335     \@preamerr \thr@@ \or
336     \@preamerr \tw@ \addtopreamble\@sharp \or
337     \or
338     \else \@preamerr \@ne \fi
339     \def\the@toks{\the\toks}

```

After an utilisation of the modified version of `\@mkpream`, we come back to the original version because there may be occurrences of the classical `{array}` in the cells of our array (of `nicematrix`).

```

340   \cs_gset_eq:NN \@mkpream \@@_old_mkpream:
341   }

```

The classes of REVTeX do their own redefinition of `\array` and that's why the previous mechanism is not compatible with REVTeX. However, it would probably be possible to do something similar for REVTeX...

```

342 \hook_gput_code:nnn { begindocument } { . }
343 {
344   \bool_if:NTF \c_@@_revtex_bool
345     { \cs_set_eq:NN \@@_redefine_mkpream: \prg_do_nothing: }
346     {
347       \IfPackageLoadedTF { arydshln }
348       { \cs_set_eq:NN \@@_redefine_mkpream: \prg_do_nothing: }
349       {
350         \cs_new_protected:Npn \@@_redefine_mkpream:
351           { \cs_set_eq:NN \@mkpream \@@_mkpream: }
352       }
353     }
354 }

355 \cs_new_protected:Npn \@@_mkpream_colortbl: { }
356 \hook_gput_code:nnn { begindocument } { . }
357 {
358   \IfPackageLoadedTF { colortbl }
359   {
360     \cs_set_protected:Npn \@@_mkpream_colortbl:
361     {

```

The following lines are a patch added to `\@mkpream` by `colortbl` (by storing the version of `\@mkpream` provided by `array` in `\@mkpreamarray`). Since you do a redefinition of `\@mkpream`, you have to add the following lines in our redefinition when `colortbl` is loaded.

```

362       \cs_set_eq:NN \CT@setup \relax
363       \cs_set_eq:NN \CT@color \relax
364       \cs_set_eq:NN \CT@do@color \relax
365       \cs_set_eq:NN \color \relax
366       \cs_set_eq:NN \CT@column@color \relax
367       \cs_set_eq:NN \CT@row@color \relax
368       \cs_set_eq:NN \CT@cell@color \relax
369     }
370   }
371   { }
372 }

```

## 5 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.

```

373 \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).

```

374 \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.

```

375 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
376 { \int_use:N \g_@@_env_int }

```

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

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

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

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

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

```
380 \bool_new:N \g_@@_delims_bool
381 \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.

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

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

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

```
385 \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.

```
386 \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).

```
387 \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.).

```
388 \dim_new:N \l_@@_col_width_dim
389 \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.

```
390 \int_new:N \g_@@_row_total_int
391 \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).

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

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

```
393 \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.

```
394 \tl_new:N \l_@@_hpos_cell_tl
395 \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`.

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

Idem for the mono-row blocks.

```
397 \dim_new:N \g_@@_blocks_ht_dim
398 \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}`).

```
399 \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`.

```
400 \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.

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

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

```
402 \bool_new:N \l_@@_notes_detect_duplicates_bool
403 \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.

```
404 \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).

```
405 \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).

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

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

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

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

```
408 \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.

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

```
410 \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 content 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 }`).

```
411 \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.

```
412 \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.

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

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

```
415 \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}`).

```
416 \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`).

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

For `\multicolumn`.

```
418 \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`.

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

```
420 \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 `:`.

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

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

```
423 \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`).

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

```
425 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
426 {
```

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

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

```
429   \fi:
```

```
430 }
```

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.

```
431 \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”.

```
432 \colorlet { nicematrix-last-col } { . }
433 \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*).

```
434 \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*.

```
435 \tl_new:N \g_@@_com_or_env_str
436 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
437 \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:VnTF` and not `\tl_if_eq:NnTF` because we need to be fully expandable).

```
438 \cs_new:Npn \@@_full_name_env:
439 {
440   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
441   { command \space \c_backslash_str \g_@@_name_env_str }
442   { environment \space \{ \g_@@_name_env_str \} }
443 }
```

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

```
444 \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.

```
445 \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.

```
446 \tl_new:N \g_@@_pre_code_before_tl
447 \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.

```
448 \tl_new:N \g_@@_pre_code_after_tl
449 \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.

```
450 \bool_new:N \l_@@_in_code_after_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.

```
451 \int_new:N \l_@@_old_iRow_int
452 \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).

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

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

```
454 \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.

```
455 \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.

```
456 \bool_new:N \l_@@_X_columns_aux_bool
```

```
457 \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.

```
458 \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).

```
459 \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).

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

`\l_@@_code_before_tl` may contain two types of informations:

- A `code-before` written in the `aux` file by a previous run. When the `aux` file is read, this `code-before` is stored in `\g_@@_code_before_i_tl` (where  $i$  is the number of the environment) and, at the beginning of the environment, it will be put in `\l_@@_code_before_tl`.
- The final user can explicitly add material in `\l_@@_code_before_tl` by using the key `code-before` or the keyword `\CodeBefore` (with the keyword `\Body`).

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

```
462 \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).

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

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

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

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

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

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

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

```
468 \dim_new:N \l_@@_tmpc_dim
469 \dim_new:N \l_@@_tmpd_dim
```

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

```
470 \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”.

```
471 \dim_new:N \g_@@_width_last_col_dim
472 \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.

```
473 \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!).

```
474 \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}`.

```
475 \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.

```
476 \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 sequence.

```
477 \seq_new:N \l_@@_corners_cells_seq
```

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`).

```
478 \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.

```
479 \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  $n$ ) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
480 \seq_new:N \g_@@_multicolumn_cells_seq
481 \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).

```
482 \int_new:N \l_@@_row_min_int
483 \int_new:N \l_@@_row_max_int
484 \int_new:N \l_@@_col_min_int
485 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
486 \int_new:N \l_@@_start_int
487 \int_set_eq:NN \l_@@_start_int \c_one_int
488 \int_new:N \l_@@_end_int
489 \int_new:N \l_@@_local_start_int
490 \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.

```
491 \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).

```
492 \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`.

```
493 \tl_new:N \l_@@_fill_tl
494 \tl_new:N \l_@@_opacity_tl
495 \tl_new:N \l_@@_draw_tl
496 \seq_new:N \l_@@_tikz_seq
497 \clist_new:N \l_@@_borders_clist
498 \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.

```
499 \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`.

```
500 \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.

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

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

```
502 \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`).

```
503 \str_new:N \l_@@_hpos_block_str
504 \str_set:Nn \l_@@_hpos_block_str { c }
505 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

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

```
506 \bool_new:N \l_@@_nocolor_used_bool
```

For the vertical position, the possible values are c, t and b.

```
507 \str_new:N \l_@@_vpos_block_str
508 \str_set:Nn \l_@@_vpos_block_str { c }
```

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

```
509 \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).

```
510 \bool_new:N \l_@@_vlines_block_bool
511 \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.

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

513 \dim_new:N \l_@@_submatrix_extra_height_dim
514 \dim_new:N \l_@@_submatrix_left_xshift_dim
515 \dim_new:N \l_@@_submatrix_right_xshift_dim
516 \clist_new:N \l_@@_hlines_clist
517 \clist_new:N \l_@@_vlines_clist
518 \clist_new:N \l_@@_submatrix_hlines_clist
519 \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}).

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

The following flag will be used by (for instance) \l\_@@\_vline\_ii:. When \l\_@@\_dotted\_bool is true, a dotted line (with our system) will be drawn.

```
521 \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).

```
522 \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.

```
523 \int_new:N \l_@@_first_row_int
524 \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.

```
525 \int_new:N \l_@@_first_col_int
526 \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).

```
527 \int_new:N \l_@@_last_row_int
528 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
530 \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`.

```
531 \int_new:N \l_@@_last_col_int
532 \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.

```
533 \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`).

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

## Some utilities

```
535 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
536 {
537   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
538   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
539 }
```

---

<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.

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.

```

540 \cs_new_protected:Npn \@@_expand_clist:N #1
541 {
542   \clist_if_in:NVF #1 \c_@@_all_tl
543   {
544     \clist_clear:N \l_tmpa_clist
545     \clist_map_inline:Nn #1
546     {
547       \tl_if_in:nnTF { ##1 } { - }
548       { \@@_cut_on_hyphen:w ##1 \q_stop }
549       {
550         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
551         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
552       }
553       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
554       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
555     }
556     \tl_set_eq:NN #1 \l_tmpa_clist
557   }
558 }

```

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.

```

559 \hook_gput_code:nnn { begindocument } { . }
560 {
561   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
562   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
563   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
564 }

```

## 6 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 tabulernote}`. The first component is the optional argument (between square brackets) of the command `\tabulernote` (if the optional argument is not used, the value will be the special marker `\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 `tabulernote` will be used to count the tabular notes during the construction of the array (this counter won't be used during the composition of the notes at the end of the array). You use a LaTeX counter because we will use `\refstepcounter` in order to have the tabular notes referenceable.

```

565 \newcounter { tabulernote }
566 \seq_new:N \g_@@_notes_seq
567 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

568 \tl_new:N \g_@@_tabulernote_tl

```

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

```

569 \seq_new:N \l_@@_notes_labels_seq
570 \newcounter{nicematrix_draft}
571 \cs_new_protected:Npn \@@_notes_format:n #1
572 {
573   \setcounter { nicematrix_draft } { #1 }
574   \@@_notes_style:n { nicematrix_draft }
575 }

```

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

```

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

```

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

```

577 \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`.

```

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

```

We define `\thetabulernote` 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`.

```

579 \cs_set:Npn \thetabulernote { { \@@_notes_style:n { tabulernote } } }

```

---

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

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`).

```

580 \hook_gput_code:nnn { begindocument } { . }
581 {
582   \IfPackageLoadedTF { enumitem }
583   {

```

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.

```

584     \newlist { tabularnotes } { enumerate } { 1 }
585     \setlist [ tabularnotes ]
586     {
587       topsep = 0pt ,
588       noitemsep ,
589       leftmargin = * ,
590       align = left ,
591       labelsep = 0pt ,
592       label =
593         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
594     }
595     \newlist { tabularnotes* } { enumerate* } { 1 }
596     \setlist [ tabularnotes* ]
597     {
598       afterlabel = \nobreak ,
599       itemjoin = \quad ,
600       label =
601         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
602     }

```

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`.

```

603     \NewDocumentCommand \tabularnote { o m }
604     {
605       \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } \l_@@_in_env_bool
606       {
607         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
608         { \@@_error:n { tabularnote~forbidden } }
609         {
610           \bool_if:NTF \l_@@_in_caption_bool
611             \@@_tabularnote_caption:nn
612             \@@_tabularnote:nn
613             { #1 } { #2 }
614         }
615       }
616     }
617   }
618   {
619     \NewDocumentCommand \tabularnote { o m }
620     {
621       \@@_error_or_warning:n { enumitem~not~loaded }
622       \@@_gredirect_none:n { enumitem~not~loaded }
623     }
624   }
625 }

626 \cs_new_protected:Npn \@@_test_first_novaluen { #1 #2 #3
627   { \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 `\c_novalue_tl`) and #2 is the mandatory argument of `\tabularnote`.

```
628 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
629 {
```

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).

```
630 \int_zero:N \l_tmpa_int
631 \bool_if:NT \l_@@_notes_detect_duplicates_bool
632 {
```

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 `\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`.

```
633 \int_zero:N \l_tmpb_int
634 \seq_map_indexed_inline:Nn \g_@@_notes_seq
635 {
636   \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
637   \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
638   {
639     \tl_if_novalue:nTF { #1 }
640     { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
641     { \int_set:Nn \l_tmpa_int { ##1 } }
642     \seq_map_break:
643   }
644 }
645 \int_if_zero:nF \l_tmpa_int
646 { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
647 }
648 \int_if_zero:nT \l_tmpa_int
649 {
650   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
651   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
652 }
653 \seq_put_right:Nx \l_@@_notes_labels_seq
654 {
655   \tl_if_novalue:nTF { #1 }
656   {
657     \@@_notes_format:n
658     {
659       \int_eval:n
660       {
661         \int_if_zero:nTF \l_tmpa_int
662         \c@tabularnote
663         \l_tmpa_int
664       }
665     }
666   }
667   { #1 }
668 }
669 \peek_meaning:NF \tabularnote
670 {
```

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`.

```
671 \hbox_set:Nn \l_tmpa_box
672 {
```

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

```
673 \@@_notes_label_in_tabular:n
674 {
675 \seq_use:Nnnn
676 \l_@@_notes_labels_seq { , } { , } { , }
677 }
678 }
```

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

```
679 \int_gdecr:N \c@tabularnote
680 \int_set_eq:NN \l_tmpa_int \c@tabularnote
681 \refstepcounter { tabularnote }
682 \int_compare:nNnT \l_tmpa_int = \c@tabularnote
683 { \int_gincr:N \c@tabularnote }
684 \seq_clear:N \l_@@_notes_labels_seq
685 \bool_lazy_or:nnTF
686 { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
687 { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
688 {
689 \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).

```
690 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
691 }
692 { \box_use:N \l_tmpa_box }
693 }
694 }
```

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`.

```
695 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
696 {
697 \bool_if:NTF \g_@@_caption_finished_bool
698 {
699 \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
700 { \int_gzero:N \c@tabularnote }
```

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

```
701 \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
702 { \@@_error:n { Identical~notes~in~caption } }
703 }
704 {
```

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

```
705 \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
706 {
```

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

```

707         \bool_gset_true:N \g_@@_caption_finished_bool
708         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
709         \int_gzero:N \c@tabularnote
710     }
711     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
712 }

```

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!

```

713     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
714     \seq_put_right:Nx \l_@@_notes_labels_seq
715     {
716         \tl_if_novalue:nTF { #1 }
717         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
718         { #1 }
719     }
720     \peek_meaning:NF \tabularnote
721     {
722         \@@_notes_label_in_tabular:n
723         { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
724         \seq_clear:N \l_@@_notes_labels_seq
725     }
726 }

727 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
728 { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }

```

## 7 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.

```

729 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
730 {
731     \begin { pgfscope }
732     \pgfset
733     {
734         inner~sep = \c_zero_dim ,
735         minimum~size = \c_zero_dim
736     }
737     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
738     \pgfnode
739     { rectangle }
740     { center }
741     {
742         \vbox_to_ht:nn
743         { \dim_abs:n { #5 - #3 } }
744         {
745             \vfill
746             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
747         }
748     }
749     { #1 }
750     { }
751     \end { pgfscope }
752 }

```

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.

```

753 \cs_new_protected:Npn \@@pgf_rect_node:nnn #1 #2 #3
754 {
755   \begin { pgfscope }
756   \pgfset
757   {
758     inner~sep = \c_zero_dim ,
759     minimum~size = \c_zero_dim
760   }
761   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
762   \pgfpointdiff { #3 } { #2 }
763   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
764   \pgfnode
765   { rectangle }
766   { center }
767   {
768     \vbox_to_ht:nn
769     { \dim_abs:n \l_tmpb_dim }
770     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
771   }
772   { #1 }
773   { }
774   \end { pgfscope }
775 }
```

## 8 The options

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

```

776 \tl_new:N \l_@@_caption_tl
777 \tl_new:N \l_@@_short_caption_tl
778 \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).

```

779 \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.

```

780 \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`.

```

781 \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`).

```

782 \dim_new:N \l_@@_cell_space_top_limit_dim
783 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
784 \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.

```
785 \dim_new:N \l_@@_xdots_inter_dim
786 \hook_gput_code:nnn { begindocument } { . }
787 { \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).

```
788 \dim_new:N \l_@@_xdots_shorten_start_dim
789 \dim_new:N \l_@@_xdots_shorten_end_dim
790 \hook_gput_code:nnn { begindocument } { . }
791 {
792   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
793   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
794 }
```

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.

```
795 \dim_new:N \l_@@_xdots_radius_dim
796 \hook_gput_code:nnn { begindocument } { . }
797 { \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.

```
798 \tl_new:N \l_@@_xdots_line_style_tl
799 \tl_const:Nn \c_@@_standard_tl { standard }
800 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax`.

```
801 \bool_new:N \l_@@_light_syntax_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).

```
802 \tl_new:N \l_@@_baseline_tl
803 \tl_set:Nn \l_@@_baseline_tl { c }
```

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`).

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

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

```
805 \bool_new:N \l_@@_parallelize_diags_bool
806 \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.

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

```
808 \dim_new:N \l_@@_notes_above_space_dim
809 \hook_gput_code:nnn { begindocument } { . }
810 { \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.).

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

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

```
812 \cs_new_protected:Npn \@@_reset_arraystretch:
813 { \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).

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

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

```
815 \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.

```
816 \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”.

```
817 \bool_new:N \l_@@_medium_nodes_bool
818 \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).

```
819 \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`).

```
820 \dim_new:N \l_@@_left_margin_dim
821 \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`.

```
822 \dim_new:N \l_@@_extra_left_margin_dim
823 \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.

```
824 \tl_new:N \l_@@_end_of_row_tl
825 \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 “:”.

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

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

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

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is fonction of its size. That’s why we create an option called `delimiters/max-width` which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

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

```
829 \keys_define:nn { NiceMatrix / xdots }
830 {
831   shorten-start .code:n =
832     \hook_gput_code:nnn { begindocument } { . }
833     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
834   shorten-end .code:n =
835     \hook_gput_code:nnn { begindocument } { . }
836     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
837   shorten-start .value_required:n = true ,
838   shorten-end .value_required:n = true ,
839   shorten .code:n =
840     \hook_gput_code:nnn { begindocument } { . }
841     {
842       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
843       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
844     } ,
845   shorten .value_required:n = true ,
846   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
847   horizontal-labels .default:n = true ,
848   line-style .code:n =
849     {
850       \bool_lazy_or:nnTF
851         { \cs_if_exist_p:N \tikzpicture }
852         { \str_if_eq_p:nn { #1 } { standard } }
853         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
854         { \@@_error:n { bad~option~for~line~style } }
855     } ,
856   line-style .value_required:n = true ,
857   color .tl_set:N = \l_@@_xdots_color_tl ,
858   color .value_required:n = true ,
859   radius .code:n =
860     \hook_gput_code:nnn { begindocument } { . }
861     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
862   radius .value_required:n = true ,
863   inter .code:n =
864     \hook_gput_code:nnn { begindocument } { . }
865     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
866   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 a absent `^{\dots}`.

```
867   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
868   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
869   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).

```

870     draw-first .code:n = \prg_do_nothing: ,
871     unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
872 }

```

```

873 \keys_define:nn { NiceMatrix / rules }
874 {
875     color .tl_set:N = \l_@@_rules_color_tl ,
876     color .value_required:n = true ,
877     width .dim_set:N = \arrayrulewidth ,
878     width .value_required:n = true ,
879     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
880 }

```

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

```

881 \keys_define:nn { NiceMatrix / Global }
882 {
883     no-cell-nodes .code:n =
884         \cs_set_protected:Npn \@@_node_for_cell:
885         { \box_use_drop:N \l_@@_cell_box } ,
886     no-cell-nodes .value_forbidden:n = true ,
887     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
888     rounded-corners .default:n = 4 pt ,
889     custom-line .code:n = \@@_custom_line:n { #1 } ,
890     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
891     rules .value_required:n = true ,
892     standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
893     standard-cline .default:n = true ,
894     cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
895     cell-space-top-limit .value_required:n = true ,
896     cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
897     cell-space-bottom-limit .value_required:n = true ,
898     cell-space-limits .meta:n =
899     {
900         cell-space-top-limit = #1 ,
901         cell-space-bottom-limit = #1 ,
902     } ,
903     cell-space-limits .value_required:n = true ,
904     xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
905     light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
906     light-syntax .default:n = true ,
907     end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
908     end-of-row .value_required:n = true ,
909     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
910     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
911     last-row .int_set:N = \l_@@_last_row_int ,
912     last-row .default:n = -1 ,
913     code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
914     code-for-first-col .value_required:n = true ,
915     code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
916     code-for-last-col .value_required:n = true ,
917     code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
918     code-for-first-row .value_required:n = true ,
919     code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
920     code-for-last-row .value_required:n = true ,
921     hlines .clist_set:N = \l_@@_hlines_clist ,
922     vlines .clist_set:N = \l_@@_vlines_clist ,
923     hlines .default:n = all ,

```

```

924 vlines .default:n = all ,
925 vlines-in-sub-matrix .code:n =
926 {
927     \tl_if_single_token:nTF { #1 }
928     {
929         \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
930         { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

931     { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
932 }
933 { \@@_error:n { One~letter-allowed } }
934 } ,
935 vlines-in-sub-matrix .value_required:n = true ,
936 hvlines .code:n =
937 {
938     \bool_set_true:N \l_@@_hvlines_bool
939     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
940     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
941 } ,
942 hvlines-except-borders .code:n =
943 {
944     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
945     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
946     \bool_set_true:N \l_@@_hvlines_bool
947     \bool_set_true:N \l_@@_except_borders_bool
948 } ,
949 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.

```

950 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
951 renew-dots .value_forbidden:n = true ,
952 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
953 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
954 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
955 create-extra-nodes .meta:n =
956 { create-medium-nodes , create-large-nodes } ,
957 left-margin .dim_set:N = \l_@@_left_margin_dim ,
958 left-margin .default:n = \arraycolsep ,
959 right-margin .dim_set:N = \l_@@_right_margin_dim ,
960 right-margin .default:n = \arraycolsep ,
961 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
962 margin .default:n = \arraycolsep ,
963 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
964 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
965 extra-margin .meta:n =
966 { extra-left-margin = #1 , extra-right-margin = #1 } ,
967 extra-margin .value_required:n = true ,
968 respect-arraystretch .code:n =
969     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
970 respect-arraystretch .value_forbidden:n = true ,
971 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
972 pgf-node-code .value_required:n = true
973 }

```

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

```

974 \keys_define:nn { NiceMatrix / Env }
975 {
976     corners .clist_set:N = \l_@@_corners_clist ,
977     corners .default:n = { NW , SW , NE , SE } ,

```

```

978 code-before .code:n =
979 {
980   \tl_if_empty:nF { #1 }
981   {
982     \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
983     \bool_set_true:N \l_@@_code_before_bool
984   }
985 } ,
986 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}`.

```

987 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
988 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
989 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
990 baseline .tl_set:N = \l_@@_baseline_tl ,
991 baseline .value_required:n = true ,
992 columns-width .code:n =
993   \tl_if_eq:nnTF { #1 } { auto }
994   { \bool_set_true:N \l_@@_auto_columns_width_bool }
995   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
996 columns-width .value_required:n = true ,
997 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.

```

998 \legacy_if:nF { measuring@ }
999 {
1000   \str_set:Nx \l_tmpa_str { #1 }
1001   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
1002   { \@@_error:nn { Duplicate~name } { #1 } }
1003   { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
1004   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
1005 } ,
1006 name .value_required:n = true ,
1007 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1008 code-after .value_required:n = true ,
1009 color-inside .code:n =
1010   \bool_set_true:N \l_@@_color_inside_bool
1011   \bool_set_true:N \l_@@_code_before_bool ,
1012 color-inside .value_forbidden:n = true ,
1013 colortbl-like .meta:n = color-inside
1014 }
1015 \keys_define:nn { NiceMatrix / notes }
1016 {
1017   para .bool_set:N = \l_@@_notes_para_bool ,
1018   para .default:n = true ,
1019   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1020   code-before .value_required:n = true ,
1021   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1022   code-after .value_required:n = true ,
1023   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1024   bottomrule .default:n = true ,
1025   style .cs_set:Np = \@@_notes_style:n #1 ,
1026   style .value_required:n = true ,
1027   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1028   label-in-tabular .value_required:n = true ,
1029   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1030   label-in-list .value_required:n = true ,
1031   enumitem-keys .code:n =
1032   {
1033     \hook_gput_code:nnn { begindocument } { . }
1034     {

```

```

1035         \IfPackageLoadedTF { enumitem }
1036         { \setlist* [ tabularnotes ] { #1 } }
1037         { }
1038     }
1039 },
1040 enumitem-keys .value_required:n = true ,
1041 enumitem-keys-para .code:n =
1042 {
1043     \hook_gput_code:nnn { begindocument } { . }
1044     {
1045         \IfPackageLoadedTF { enumitem }
1046         { \setlist* [ tabularnotes* ] { #1 } }
1047         { }
1048     }
1049 },
1050 enumitem-keys-para .value_required:n = true ,
1051 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1052 detect-duplicates .default:n = true ,
1053 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1054 }
1055 \keys_define:nn { NiceMatrix / delimiters }
1056 {
1057     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1058     max-width .default:n = true ,
1059     color .tl_set:N = \l_@@_delimiters_color_tl ,
1060     color .value_required:n = true ,
1061 }

```

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

```

1062 \keys_define:nn { NiceMatrix }
1063 {
1064     NiceMatrixOptions .inherit:n =
1065     { NiceMatrix / Global } ,
1066     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1067     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1068     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1069     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1070     SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1071     CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1072     CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1073     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1074     NiceMatrix .inherit:n =
1075     {
1076         NiceMatrix / Global ,
1077         NiceMatrix / Env ,
1078     } ,
1079     NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1080     NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1081     NiceTabular .inherit:n =
1082     {
1083         NiceMatrix / Global ,
1084         NiceMatrix / Env
1085     } ,
1086     NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1087     NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1088     NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1089     NiceArray .inherit:n =
1090     {
1091         NiceMatrix / Global ,
1092         NiceMatrix / Env ,
1093     } ,

```

```

1094   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1095   NiceArray / rules .inherit:n = NiceMatrix / rules ,
1096   pNiceArray .inherit:n =
1097   {
1098       NiceMatrix / Global ,
1099       NiceMatrix / Env ,
1100   } ,
1101   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1102   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1103 }

```

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

```

1104 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1105 {
1106   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1107   delimiters / color .value_required:n = true ,
1108   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1109   delimiters / max-width .default:n = true ,
1110   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1111   delimiters .value_required:n = true ,
1112   width .dim_set:N = \l_@@_width_dim ,
1113   width .value_required:n = true ,
1114   last-col .code:n =
1115     \tl_if_empty:nF { #1 }
1116     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1117     \int_zero:N \l_@@_last_col_int ,
1118   small .bool_set:N = \l_@@_small_bool ,
1119   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.

```

1120   renew-matrix .code:n = \@@_renew_matrix: ,
1121   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}`.

```

1122   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.

```

1123   columns-width .code:n =
1124     \tl_if_eq:nnTF { #1 } { auto }
1125     { \@@_error:n { Option~auto~for~columns-width } }
1126     { \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.

```

1127   allow-duplicate-names .code:n =
1128     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1129   allow-duplicate-names .value_forbidden:n = true ,
1130   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1131   notes .value_required:n = true ,
1132   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1133   sub-matrix .value_required:n = true ,
1134   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1135   matrix / columns-type .value_required:n = true ,
1136   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1137   caption-above .default:n = true ,
1138   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1139 }

```

`\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.

```
1140 \NewDocumentCommand \NiceMatrixOptions { m }
1141 { \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.

```
1142 \keys_define:nn { NiceMatrix / NiceMatrix }
1143 {
1144   last-col .code:n = \tl_if_empty:nTF { #1 }
1145               {
1146                 \bool_set_true:N \l_@@_last_col_without_value_bool
1147                 \int_set:Nn \l_@@_last_col_int { -1 }
1148               }
1149               { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1150   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1151   columns-type .value_required:n = true ,
1152   l .meta:n = { columns-type = l } ,
1153   r .meta:n = { columns-type = r } ,
1154   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1155   delimiters / color .value_required:n = true ,
1156   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1157   delimiters / max-width .default:n = true ,
1158   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1159   delimiters .value_required:n = true ,
1160   small .bool_set:N = \l_@@_small_bool ,
1161   small .value_forbidden:n = true ,
1162   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1163 }
```

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

```
1164 \keys_define:nn { NiceMatrix / NiceArray }
1165 {
```

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.

```
1166   small .bool_set:N = \l_@@_small_bool ,
1167   small .value_forbidden:n = true ,
1168   last-col .code:n = \tl_if_empty:nF { #1 }
1169                       { \@@_error:n { last-col-non-empty-for-NiceArray } }
1170                       \int_zero:N \l_@@_last_col_int ,
1171   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1172   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1173   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1174 }

1175 \keys_define:nn { NiceMatrix / pNiceArray }
1176 {
1177   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1178   last-col .code:n = \tl_if_empty:nF { #1 }
1179                       { \@@_error:n { last-col-non-empty-for-NiceArray } }
1180                       \int_zero:N \l_@@_last_col_int ,
1181   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1182   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1183   delimiters / color .value_required:n = true ,
1184   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1185   delimiters / max-width .default:n = true ,
1186   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1187   delimiters .value_required:n = true ,
1188   small .bool_set:N = \l_@@_small_bool ,
1189   small .value_forbidden:n = true ,
```

```

1190   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1191   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1192   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1193 }

```

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

```

1194 \keys_define:nn { NiceMatrix / NiceTabular }
1195 {

```

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.

```

1196   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1197   \bool_set_true:N \l_@@_width_used_bool ,
1198   width .value_required:n = true ,
1199   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1200   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1201   tabularnote .value_required:n = true ,
1202   caption .tl_set:N = \l_@@_caption_tl ,
1203   caption .value_required:n = true ,
1204   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1205   short-caption .value_required:n = true ,
1206   label .tl_set:N = \l_@@_label_tl ,
1207   label .value_required:n = true ,
1208   last-col .code:n = \tl_if_empty:nF {#1}
1209   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1210   \int_zero:N \l_@@_last_col_int ,
1211   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1212   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1213   unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1214 }

```

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
```

```

1215 \keys_define:nn { NiceMatrix / CodeAfter }
1216 {
1217   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1218   delimiters / color .value_required:n = true ,
1219   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1220   rules .value_required:n = true ,
1221   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1222   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1223   sub-matrix .value_required:n = true ,
1224   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1225 }

```

## 9 Important code used by {NiceArrayWithDelims}

The pseudo-environment \@@\_cell\_begin:w-\@@\_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}).

```

1226 \cs_new_protected:Npn \@@_cell_begin:w
1227 {

```

`\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.

```
1228 \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).

```
1229 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1230 \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.

```
1231 \int_compare:nNnT \c@jCol = \c_one_int
1232 { \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:`.

```
1233 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1234 \@@_tuning_not_tabular_begin:
1235 \@@_tuning_first_row:
1236 \@@_tuning_last_row:
1237 \g_@@_row_style_tl
1238 }
```

The following command will be nullified unless there is a first row.

```
1239 \cs_new_protected:Npn \@@_tuning_first_row:
1240 {
1241   \int_if_zero:nT \c@iRow
1242   {
1243     \int_compare:nNnT \c@jCol > \c_zero_int
1244     {
1245       \l_@@_code_for_first_row_tl
1246       \xglobal \colorlet { nicematrix-first-row } { . }
1247     }
1248   }
1249 }
```

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

```
1250 \cs_new_protected:Npn \@@_tuning_last_row:
1251 {
1252   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1253   {
1254     \l_@@_code_for_last_row_tl
1255     \xglobal \colorlet { nicematrix-last-row } { . }
1256   }
1257 }
```

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

```
1258 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1259 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1260 {
1261   \c_math_toggle_token
```

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

```
1262 \@@_tuning_key_small:
1263 }
1264 \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.

```

1265 \cs_new_protected:Npn \@@_begin_of_row:
1266 {
1267   \int_gincr:N \c@iRow
1268   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1269   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1270   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1271   \pgfpicture
1272   \pgfrememberpicturepositiononpagetrue
1273   \pgfcoordinate
1274   { \@@_env: - row - \int_use:N \c@iRow - base }
1275   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1276   \str_if_empty:NF \l_@@_name_str
1277   {
1278     \pgfnodealias
1279     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1280     { \@@_env: - row - \int_use:N \c@iRow - base }
1281   }
1282   \endpgfpicture
1283 }

```

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.

```

1284 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1285 {
1286   \int_if_zero:nTF \c@iRow
1287   {
1288     \dim_gset:Nn \g_@@_dp_row_zero_dim
1289     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1290     \dim_gset:Nn \g_@@_ht_row_zero_dim
1291     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1292   }
1293   {
1294     \int_compare:nNnT \c@iRow = \c_one_int
1295     {
1296       \dim_gset:Nn \g_@@_ht_row_one_dim
1297       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1298     }
1299   }
1300 }

1301 \cs_new_protected:Npn \@@_rotate_cell_box:
1302 {
1303   \box_rotate:Nn \l_@@_cell_box { 90 }
1304   \bool_if:NTF \g_@@_rotate_c_bool
1305   {
1306     \hbox_set:Nn \l_@@_cell_box
1307     {
1308       \c_math_toggle_token
1309       \vcenter { \box_use:N \l_@@_cell_box }
1310       \c_math_toggle_token
1311     }
1312   }
1313   {
1314     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1315     {
1316       \vbox_set_top:Nn \l_@@_cell_box
1317       {

```

```

1318         \vbox_to_zero:n { }
1319         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1320         \box_use:N \l_@@_cell_box
1321     }
1322 }
1323 }
1324 \bool_gset_false:N \g_@@_rotate_bool
1325 \bool_gset_false:N \g_@@_rotate_c_bool
1326 }
1327 \cs_new_protected:Npn \@@_adjust_size_box:
1328 {
1329     \dim_compare:nNtT \g_@@_blocks_wd_dim > \c_zero_dim
1330     {
1331         \box_set_wd:Nn \l_@@_cell_box
1332         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1333         \dim_gzero:N \g_@@_blocks_wd_dim
1334     }
1335     \dim_compare:nNtT \g_@@_blocks_dp_dim > \c_zero_dim
1336     {
1337         \box_set_dp:Nn \l_@@_cell_box
1338         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1339         \dim_gzero:N \g_@@_blocks_dp_dim
1340     }
1341     \dim_compare:nNtT \g_@@_blocks_ht_dim > \c_zero_dim
1342     {
1343         \box_set_ht:Nn \l_@@_cell_box
1344         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1345         \dim_gzero:N \g_@@_blocks_ht_dim
1346     }
1347 }
1348 \cs_new_protected:Npn \@@_cell_end:
1349 {

```

The following command is nullified in the tabulars.

```

1350     \@@_tuning_not_tabular_end:
1351     \hbox_set_end:
1352     \@@_cell_end_i:
1353 }
1354 \cs_new_protected:Npn \@@_cell_end_i:
1355 {

```

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.

```

1356     \g_@@_cell_after_hook_tl
1357     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1358     \@@_adjust_size_box:
1359     \box_set_ht:Nn \l_@@_cell_box
1360     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1361     \box_set_dp:Nn \l_@@_cell_box
1362     { \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”).

```

1363     \@@_update_max_cell_width:

```

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

```

1364     \@@_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 very 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.

```

1365 \bool_if:NTF \g_@@_empty_cell_bool
1366 { \box_use_drop:N \l_@@_cell_box }
1367 {
1368   \bool_if:NTF \g_@@_not_empty_cell_bool
1369   \@@_node_for_cell:
1370   {
1371     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1372     \@@_node_for_cell:
1373     { \box_use_drop:N \l_@@_cell_box }
1374   }
1375 }
1376 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1377 \bool_gset_false:N \g_@@_empty_cell_bool
1378 \bool_gset_false:N \g_@@_not_empty_cell_bool
1379 }

```

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

```

1380 \cs_new_protected:Npn \@@_update_max_cell_width:
1381 {
1382   \dim_gset:Nn \g_@@_max_cell_width_dim
1383   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1384 }

```

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`).

```

1385 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1386 {
1387   \@@_math_toggle:
1388   \hbox_set_end:
1389   \bool_if:NF \g_@@_rotate_bool
1390   {
1391     \hbox_set:Nn \l_@@_cell_box
1392     {
1393       \makebox [ \l_@@_col_width_dim ] [ s ]
1394       { \hbox_unpack_drop:N \l_@@_cell_box }
1395     }
1396   }
1397   \@@_cell_end_i:
1398 }

```

```

1399 \pgfset
1400 {
1401   nicematrix / cell-node /.style =
1402   {
1403     inner-sep = \c_zero_dim ,
1404     minimum-width = \c_zero_dim

```

```

1405   }
1406 }

```

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

```

1407 \cs_new_protected:Npn \@@_node_for_cell:
1408 {
1409   \pgfpicture
1410   \pgfsetbaseline \c_zero_dim
1411   \pgfrememberpicturerepositiononpagetrue
1412   \pgfset { nicematrix / cell-node }
1413   \pgfnode
1414     { rectangle }
1415     { base }
1416   {

```

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).

```

1417     \set@color
1418     \box_use_drop:N \l_@@_cell_box
1419   }
1420   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1421   { \l_@@_pgf_node_code_tl }
1422   \str_if_empty:NF \l_@@_name_str
1423   {
1424     \pgfnodealias
1425       { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1426       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1427   }
1428   \endpgfpicture
1429 }

```

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.

```

1430 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1431 {
1432   \cs_new_protected:Npn \@@_patch_node_for_cell:
1433   {
1434     \hbox_set:Nn \l_@@_cell_box
1435     {
1436       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1437       \hbox_overlap_left:n
1438       {
1439         \pgfsys@markposition
1440         { \@@_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`, `divps`, `ps2pdf` (or Adobe Distiller). However, it seems to work.

```

1441         #1
1442       }
1443     \box_use:N \l_@@_cell_box
1444     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1445     \hbox_overlap_left:n
1446     {
1447       \pgfsys@markposition
1448       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1449     }
1450   }
1451 }
1452 }
1453 }

```

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

```

1454 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1455 {
1456   \@@_patch_node_for_cell:n
1457     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1458 }
1459 { \@@_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).

```

1460 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1461 {
1462   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1463     { g_@@_#2 _ lines _ tl }
1464   {
1465     \use:c { @@ _ draw _ #2 : nnn }
1466     { \int_use:N \c@iRow }
1467     { \int_use:N \c@jCol }
1468     { \exp_not:n { #3 } }
1469   }
1470 }

```

```

1471 \cs_new_protected:Npn \@@_array:
1472 {

```

The following line is only a speed-up: it's a redefinition of `\@mkpream` of `array` in order to speed up the compilation by deleting one line of code in `\@mkpream` (the expansion of the preamble). In the classes of REVTeX, that command `\@@_redefine_mkpream:` will be nullified (no speed-up).

```

1473 \@@_redefine_mkpream:
1474 \dim_set:Nn \col@sep
1475 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1476 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1477 { \cs_set_nopar:Npn \@halignto { } }
1478 { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1479 \@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:VnTF` is fully expandable and we need something fully expandable here.

```

1480 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1481 }

```

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.

```
1482 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1483 \cs_new_protected:Npn \@@_create_row_node:
1484 {
1485   \int_compare:nNtT \c@iRow > \g_@@_last_row_node_int
1486   {
1487     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1488     \@@_create_row_node_i:
1489   }
1490 }

1491 \cs_new_protected:Npn \@@_create_row_node_i:
1492 {
```

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

```
1493   \hbox
1494   {
1495     \bool_if:NT \l_@@_code_before_bool
1496     {
1497       \vtop
1498       {
1499         \skip_vertical:N 0.5\arrayrulewidth
1500         \pgfsys@markposition
1501         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1502         \skip_vertical:N -0.5\arrayrulewidth
1503       }
1504     }
1505     \pgfpicture
1506     \pgfrememberpicturepositiononpagetrue
1507     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1508     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1509     \str_if_empty:NF \l_@@_name_str
1510     {
1511       \pgfnodealias
1512       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1513       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1514     }
1515     \endpgfpicture
1516   }
1517 }
```

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

```
1518 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1519 \cs_new_protected:Npn \@@_everycr_i:
1520 {
1521   \int_gzero:N \c@jCol
1522   \bool_gset_false:N \g_@@_after_col_zero_bool
1523   \bool_if:NF \g_@@_row_of_col_done_bool
1524   {
1525     \@@_create_row_node:
```

We don't draw now the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for theses rules (the rules will be drawn by PGF).

```
1526   \tl_if_empty:NF \l_@@_hlines_clist
1527   {
1528     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1529     {
1530       \exp_args:NNe
1531       \clist_if_in:NnT
1532       \l_@@_hlines_clist
```

```

1533         { \int_eval:n { \c@iRow + 1 } }
1534     }
1535 }

```

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.

```

1536     \int_compare:nNnT \c@iRow > { -1 }
1537     {
1538         \int_compare:nNnF \c@iRow = \l_@@_last_row_int

```

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. We use a TeX group in order to limit the scope of `\CT@arc@`.

```

1539         { \hrule height \arrayrulewidth width \c_zero_dim }
1540     }
1541 }
1542 }
1543 }
1544 }

```

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

```

1545 \cs_set_protected:Npn \@@_renew_dots:
1546 {
1547     \cs_set_eq:NN \ldots \@@_Ldots
1548     \cs_set_eq:NN \cdots \@@_Cdots
1549     \cs_set_eq:NN \vdots \@@_Vdots
1550     \cs_set_eq:NN \ddots \@@_Ddots
1551     \cs_set_eq:NN \iddots \@@_Iddots
1552     \cs_set_eq:NN \dots \@@_Ldots
1553     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1554 }
1555 \cs_new_protected:Npn \@@_test_color_inside:
1556 {
1557     \bool_if:NF \l_@@_color_inside_bool
1558     {

```

We will issue an error only during the first run.

```

1559         \bool_if:NF \g_@@_aux_found_bool
1560         { \@@_error:n { without~color~inside } }
1561     }
1562 }

```

```

1563 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1564 \hook_gput_code:nnn { begindocument } { . }
1565 {
1566     \IfPackageLoadedTF { colortbl }
1567     {
1568         \cs_set_protected:Npn \@@_redefine_everycr:
1569         {
1570             \CT@everycr
1571             {
1572                 \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1573                 \@@_everycr:
1574             }
1575         }
1576     }
1577 }
1578 }

```

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>.

```

1579 \hook_gput_code:nnn { begindocument } { . }
1580 {
1581   \IfPackageLoadedTF { booktabs }
1582   {
1583     \cs_new_protected:Npn \@@_patch_booktabs:
1584       { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1585   }
1586   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1587 }

```

The following code `\@@_pre_array_ii:` is used in `{NiceArrayWithDelims}`. It exists as a standalone macro only for legibility.

```

1588 \cs_new_protected:Npn \@@_pre_array_ii:
1589 {

```

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

```

1590   \int_gzero:N \g_@@_total_X_weight_int
1591   \@@_expand_clist:N \l_@@_hlines_clist
1592   \@@_expand_clist:N \l_@@_vlines_clist
1593   \@@_patch_booktabs:
1594   \box_clear_new:N \l_@@_cell_box
1595   \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}`).

```

1596   \bool_if:NT \l_@@_small_bool
1597   {
1598     \cs_set_nopar:Npn \arraystretch { 0.47 }
1599     \dim_set:Nn \arraycolsep { 1.45 pt }

```

By default, `\@@_small_scriptstyle:` is null.

```

1600     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1601   }

1602   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1603   {
1604     \tl_put_right:Nn \@@_begin_of_row:
1605     {
1606       \pgfsys@markposition
1607       { \@@_env: - row - \int_use:N \c@iRow - base }
1608     }
1609   }

```

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`.

```

1610   \cs_set_nopar:Npn \ialign
1611   {
1612     \@@_redefine_everycr:
1613     \tabskip = \c_zero_skip

```

---

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

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`.

```

1614      \dim_gzero_new:N \g_@@_dp_row_zero_dim
1615      \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1616      \dim_gzero_new:N \g_@@_ht_row_zero_dim
1617      \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1618      \dim_gzero_new:N \g_@@_ht_row_one_dim
1619      \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1620      \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1621      \dim_gzero_new:N \g_@@_ht_last_row_dim
1622      \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1623      \dim_gzero_new:N \g_@@_dp_last_row_dim
1624      \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }

```

After its first use, the definition of `\ialign` will revert automatically to its default definition. With this programming, we will have, in the cells of the array, a clean version of `\ialign`.

```

1625      \cs_set_eq:NN \ialign \@_old_ialign:
1626      \halign
1627  }

```

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).

```

1628      \cs_set_eq:NN \@_old_ldots \ldots
1629      \cs_set_eq:NN \@_old_cdots \cdots
1630      \cs_set_eq:NN \@_old_vdots \vdots
1631      \cs_set_eq:NN \@_old_ddots \ddots
1632      \cs_set_eq:NN \@_old_iddots \iddots
1633      \bool_if:NTF \l_@@_standard_cline_bool
1634      { \cs_set_eq:NN \cline \@_standard_cline }
1635      { \cs_set_eq:NN \cline \@_cline }
1636      \cs_set_eq:NN \Ldots \@_Ldots
1637      \cs_set_eq:NN \Cdots \@_Cdots
1638      \cs_set_eq:NN \Vdots \@_Vdots
1639      \cs_set_eq:NN \Ddots \@_Ddots
1640      \cs_set_eq:NN \Iddots \@_Iddots
1641      \cs_set_eq:NN \Hline \@_Hline:
1642      \cs_set_eq:NN \Hspace \@_Hspace:
1643      \cs_set_eq:NN \Hdotsfor \@_Hdotsfor:
1644      \cs_set_eq:NN \Vdotsfor \@_Vdotsfor:
1645      \cs_set_eq:NN \Block \@_Block:
1646      \cs_set_eq:NN \rotate \@_rotate:
1647      \cs_set_eq:NN \OnlyMainNiceMatrix \@_OnlyMainNiceMatrix:n
1648      \cs_set_eq:NN \dotfill \@_dotfill:
1649      \cs_set_eq:NN \CodeAfter \@_CodeAfter:
1650      \cs_set_eq:NN \diagbox \@_diagbox:nn
1651      \cs_set_eq:NN \NotEmpty \@_NotEmpty:
1652      \cs_set_eq:NN \RowStyle \@_RowStyle:n
1653      \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1654      { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1655      \cs_set_eq:NN \cellcolor \@_cellcolor_tabular
1656      \cs_set_eq:NN \rowcolor \@_rowcolor_tabular
1657      \cs_set_eq:NN \rowcolors \@_rowcolors_tabular
1658      \cs_set_eq:NN \rowlistcolors \@_rowlistcolors_tabular
1659      \int_compare:nNt \l_@@_first_row_int > \c_zero_int

```

---

<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}`.

```

1660     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1661 \int_compare:nNt \l_@@_last_row_int < \c_zero_int
1662     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1663 \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.

```

1664 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1665 \hook_gput_code:nnn { env / tabular / begin } { . }
1666     { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1667 \@@_revert_colortbl:

```

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

```

1668 \tl_if_exist:NT \l_@@_note_in_caption_tl
1669 {
1670     \tl_if_empty:NF \l_@@_note_in_caption_tl
1671     {
1672         \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1673         \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1674     }
1675 }

```

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).

```

1676 \seq_gclear:N \g_@@_multicolumn_cells_seq
1677 \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).

```

1678 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number of rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1679 \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:w` executed at the beginning of each cell.

```

1680 \int_gzero_new:N \g_@@_col_total_int
1681 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1682 \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.

```

1683 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1684 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1685 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1686 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1687 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1688 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1689 \tl_gclear:N \g_nicematrix_code_before_tl
1690 \tl_gclear:N \g_@@_pre_code_before_tl
1691 }

```

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

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

```

1692 \cs_new_protected:Npn \@@_pre_array:
1693 {
1694   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1695   \int_gzero_new:N \c@iRow
1696   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1697   \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).

```

1698   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1699   {
1700     \bool_set_true:N \l_@@_last_row_without_value_bool
1701     \bool_if:NT \g_@@_aux_found_bool
1702       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1703   }
1704   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1705   {
1706     \bool_if:NT \g_@@_aux_found_bool
1707       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1708   }

```

If there is an exterior row, we patch a command used in `\@@_cell_begin:w` in order to keep track of some dimensions needed to the construction of that “last row”.

```

1709   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1710   {
1711     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1712       {
1713         \dim_gset:Nn \g_@@_ht_last_row_dim
1714           { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1715         \dim_gset:Nn \g_@@_dp_last_row_dim
1716           { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1717       }
1718   }

1719   \seq_gclear:N \g_@@_cols_vlism_seq
1720   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1721   \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.

```

1722   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1723   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1724   \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.

```
1725 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value  $-2$  is important.

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

```
1726 \@@_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.

```
1727 \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]`).

```
1728 \dim_zero_new:N \l_@@_left_delim_dim
1729 \dim_zero_new:N \l_@@_right_delim_dim
1730 \bool_if:NTF \g_@@_delims_bool
1731 {
```

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

```
1732 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1733 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1734 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1735 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1736 }
1737 {
1738 \dim_gset:Nn \l_@@_left_delim_dim
1739 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1740 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1741 }
```

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).

```
1742 \hbox_set:Nw \l_@@_the_array_box
1743 \skip_horizontal:N \l_@@_left_margin_dim
1744 \skip_horizontal:N \l_@@_extra_left_margin_dim
1745 \c_math_toggle_token
1746 \bool_if:NTF \l_@@_light_syntax_bool
1747 { \use:c { @@-light-syntax } }
1748 { \use:c { @@-normal-syntax } }
1749 }
```

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

```
1750 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1751 {
1752 \tl_set:Nn \l_tmpa_tl { #1 }
1753 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1754 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1755 \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1756 \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...

```
1757 \@@_pre_array:
1758 }
```

## 10 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 only for legibility).

```
1759 \cs_new_protected:Npn \@@_pre_code_before:
1760 {
```

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).

```
1761 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1762 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1763 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1764 \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.

```
1765 \pgfsys@markposition { \@@_env: - position }
1766 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1767 \pgfpicture
1768 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1769 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1770 {
1771   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1772   \pgfcoordinate { \@@_env: - row - ##1 }
1773   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1774 }
```

Now, the recreation of the `col` nodes.

```
1775 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1776 {
1777   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1778   \pgfcoordinate { \@@_env: - col - ##1 }
1779   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1780 }
```

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

```
1781 \@@_create_diag_nodes:
```

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

```
1782 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1783 \endpgfpicture
```

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

```
1784 \@@_create_blocks_nodes:
1785 \IfPackageLoadedTF { tikz }
1786 {
1787   \tikzset
1788   {
1789     every-picture / .style =
1790     { overlay , name-prefix = \@@_env: - }
1791   }
1792 }
1793 { }
1794 \cs_set_eq:NN \cellcolor \@@_cellcolor
1795 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1796 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1797 \cs_set_eq:NN \rowcolor \@@_rowcolor
```

```

1798 \cs_set_eq:NN \rowcolors \@@_rowcolors
1799 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1800 \cs_set_eq:NN \arraycolor \@@_arraycolor
1801 \cs_set_eq:NN \columncolor \@@_columncolor
1802 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1803 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1804 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1805 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1806 }

```

```

1807 \cs_new_protected:Npn \@@_exec_code_before:
1808 {
1809   \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`.

```

1810 \@@_add_to_colors_seq:nn { { nocolor } } { }
1811 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1812 \group_begin:

```

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

```

1813 \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`).

```

1814 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1815 { \@@_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`.

```

1816 \exp_last_unbraced:N \@@_CodeBefore_keys:
1817 \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.

```

1818 \@@_actually_color:
1819 \l_@@_code_before_tl
1820 \q_stop
1821 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1822 \group_end:
1823 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1824 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1825 }

```

```

1826 \keys_define:nn { NiceMatrix / CodeBefore }
1827 {
1828   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1829   create-cell-nodes .default:n = true ,
1830   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1831   sub-matrix .value_required:n = true ,
1832   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1833   delimiters / color .value_required:n = true ,
1834   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1835 }

```

```

1836 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1837 {
1838   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1839   \@@_CodeBefore:w
1840 }

```

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.

```

1841 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1842 {
1843   \bool_if:NT \g_@@_aux_found_bool
1844   {
1845     \@@_pre_code_before:
1846     #1
1847   }
1848 }

```

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.

```

1849 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1850 {
1851   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1852   {
1853     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1854     \pgfcoordinate { \@@_env: - row - ##1 - base }
1855     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1856     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1857     {
1858       \cs_if_exist:cT
1859       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1860       {
1861         \pgfsys@getposition
1862         { \@@_env: - ##1 - #####1 - NW }
1863         \@@_node_position:
1864         \pgfsys@getposition
1865         { \@@_env: - ##1 - #####1 - SE }
1866         \@@_node_position_i:
1867         \@@_pgf_rect_node:nnn
1868         { \@@_env: - ##1 - #####1 }
1869         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1870         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1871       }
1872     }
1873   }
1874   \int_step_inline:nn \c@iRow
1875   {
1876     \pgfnodealias
1877     { \@@_env: - ##1 - last }
1878     { \@@_env: - ##1 - \int_use:N \c@jCol }
1879   }
1880   \int_step_inline:nn \c@jCol
1881   {
1882     \pgfnodealias
1883     { \@@_env: - last - ##1 }
1884     { \@@_env: - \int_use:N \c@iRow - ##1 }
1885   }
1886   \@@_create_extra_nodes:
1887 }

```

```

1888 \cs_new_protected:Npn \@@_create_blocks_nodes:
1889 {
1890   \pgfpicture
1891   \pgf@relevantforpicturesizefalse
1892   \pgfrememberpicturepositiononpagetrue
1893   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1894   { \@@_create_one_block_node:nnnnn ##1 }
1895   \endpgfpicture
1896 }

```

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>

```

1897 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1898 {
1899   \tl_if_empty:nF { #5 }
1900   {
1901     \@@_qpoint:n { col - #2 }
1902     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1903     \@@_qpoint:n { #1 }
1904     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1905     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1906     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1907     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1908     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1909     \@@_pgf_rect_node:nnnnn
1910     { \@@_env: - #5 }
1911     { \dim_use:N \l_tmpa_dim }
1912     { \dim_use:N \l_tmpb_dim }
1913     { \dim_use:N \l_@@_tmpc_dim }
1914     { \dim_use:N \l_@@_tmpd_dim }
1915   }
1916 }

1917 \cs_new_protected:Npn \@@_patch_for_revtext:
1918 {
1919   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1920   \cs_set_eq:NN \insert@column \insert@column@array
1921   \cs_set_eq:NN \@classx \@classx@array
1922   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1923   \cs_set_eq:NN \@arraycr \@arraycr@array
1924   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1925   \cs_set_eq:NN \array \array@array
1926   \cs_set_eq:NN \@array \@array@array
1927   \cs_set_eq:NN \@tabular \@tabular@array
1928   \cs_set_eq:NN \@mkpream \@mkpream@array
1929   \cs_set_eq:NN \endarray \endarray@array
1930   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1931   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1932 }

```

## 11 The environment `{NiceArrayWithDelims}`

```

1933 \NewDocumentEnvironment { NiceArrayWithDelims }
1934 { m m 0 { } m ! 0 { } t \CodeBefore }
1935 {

```

<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).

```

1936 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1937 \@@_provide_pgfsyspdfmark:
1938 \bool_if:NT \g_@@_footnote_bool \savenotes

```

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

```

1939 \bgroup

1940 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1941 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1942 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1943 \int_gzero:N \g_@@_block_box_int
1944 \dim_zero:N \g_@@_width_last_col_dim
1945 \dim_zero:N \g_@@_width_first_col_dim
1946 \bool_gset_false:N \g_@@_row_of_col_done_bool
1947 \str_if_empty:NT \g_@@_name_env_str
1948 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1949 \bool_if:NTF \l_@@_tabular_bool
1950 \mode_leave_vertical:
1951 \@@_test_if_math_mode:
1952 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1953 \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.

```

1954 \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.

```

1955 \cs_if_exist:NT \tikz@library@external@loaded
1956 {
1957 \tikzexternaldisable
1958 \cs_if_exist:NT \ifstandalone
1959 { \tikzset { external / optimize = false } }
1960 }

```

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

```

1961 \int_gincr:N \g_@@_env_int
1962 \bool_if:NF \l_@@_block_auto_columns_width_bool
1963 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

The sequence `\g_@@_blocks_seq` will contain the carateristics 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`).

```

1964 \seq_gclear:N \g_@@_blocks_seq
1965 \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`.

```

1966 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1967 \seq_gclear:N \g_@@_pos_of_xdots_seq
1968 \tl_gclear_new:N \g_@@_code_before_tl
1969 \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.

---

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

```

1970 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1971 {
1972   \bool_gset_true:N \g_@@_aux_found_bool
1973   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1974 }
1975 { \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.

```

1976 \tl_gclear:N \g_@@_aux_tl
1977 \tl_if_empty:NF \g_@@_code_before_tl
1978 {
1979   \bool_set_true:N \l_@@_code_before_bool
1980   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1981 }
1982 \tl_if_empty:NF \g_@@_pre_code_before_tl
1983 { \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`.

```

1984 \bool_if:NTF \g_@@_delims_bool
1985 { \keys_set:nn { NiceMatrix / pNiceArray } }
1986 { \keys_set:nn { NiceMatrix / NiceArray } }
1987 { #3 , #5 }

```

```

1988 \@@_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:`.

```

1989 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1990 }

```

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

```

1991 {
1992   \bool_if:NTF \l_@@_light_syntax_bool
1993   { \use:c { end @@-light-syntax } }
1994   { \use:c { end @@-normal-syntax } }
1995   \c_math_toggle_token
1996   \skip_horizontal:N \l_@@_right_margin_dim
1997   \skip_horizontal:N \l_@@_extra_right_margin_dim
1998   \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.

```

1999 \bool_if:NT \l_@@_width_used_bool
2000 {
2001   \int_if_zero:nT \g_@@_total_X_weight_int
2002   { \@@_error_or_warning:n { width-without-X-columns } }
2003 }

```

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  $n$ .

```

2004 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
2005 {
2006   \tl_gput_right:Nx \g_@@_aux_tl
2007   {

```

```

2008      \bool_set_true:N \l_@@_X_columns_aux_bool
2009      \dim_set:Nn \l_@@_X_columns_dim
2010      {
2011        \dim_compare:nNnTF
2012        {
2013          \dim_abs:n
2014          { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2015        }
2016        <
2017        { 0.001 pt }
2018        { \dim_use:N \l_@@_X_columns_dim }
2019        {
2020          \dim_eval:n
2021          {
2022            ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2023            / \int_use:N \g_@@_total_X_weight_int
2024            + \l_@@_X_columns_dim
2025          }
2026        }
2027      }
2028    }
2029  }

```

It 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).

```

2030    \int_compare:nNnT \l_@@_last_row_int > { -2 }
2031    {
2032      \bool_if:NF \l_@@_last_row_without_value_bool
2033      {
2034        \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2035        {
2036          \@@_error:n { Wrong-last-row }
2037          \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2038        }
2039      }
2040    }

```

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>

```

2041    \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2042    \bool_if:NnTF \g_@@_last_col_found_bool
2043    { \int_gdecr:N \c@jCol }
2044    {
2045      \int_compare:nNnT \l_@@_last_col_int > { -1 }
2046      { \@@_error:n { last-col-not-used } }
2047    }

```

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

```

2048    \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2049    \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).

```

2050    \int_if_zero:nT \l_@@_first_col_int
2051    { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2052    \bool_if:nTF { ! \g_@@_delims_bool }
2053    {

```

---

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

```

2054 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2055 \@@_use_arraybox_with_notes_c:
2056 {
2057 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2058 \@@_use_arraybox_with_notes_b:
2059 \@@_use_arraybox_with_notes:
2060 }
2061 }

```

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).

```

2062 {
2063 \int_if_zero:nTF \l_@@_first_row_int
2064 {
2065 \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2066 \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2067 }
2068 { \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>

```

2069 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2070 {
2071 \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2072 \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2073 }
2074 { \dim_zero:N \l_tmpb_dim }
2075 \hbox_set:Nn \l_tmpa_box
2076 {
2077 \c_math_toggle_token
2078 \@@_color:o \l_@@_delimiters_color_tl
2079 \exp_after:wN \left \g_@@_left_delim_tl
2080 \vcenter
2081 {

```

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.

```

2082 \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2083 \hbox
2084 {
2085 \bool_if:NTF \l_@@_tabular_bool
2086 { \skip_horizontal:N -\tabcolsep }
2087 { \skip_horizontal:N -\arraycolsep }
2088 \@@_use_arraybox_with_notes_c:
2089 \bool_if:NTF \l_@@_tabular_bool
2090 { \skip_horizontal:N -\tabcolsep }
2091 { \skip_horizontal:N -\arraycolsep }
2092 }

```

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

```

2093 \skip_vertical:N -\l_tmpb_dim
2094 \skip_vertical:N \arrayrulewidth
2095 }
2096 \exp_after:wN \right \g_@@_right_delim_tl
2097 \c_math_toggle_token
2098 }

```

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.

```

2099 \bool_if:NTF \l_@@_delimiters_max_width_bool
2100 {

```

---

<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).

```

2101         \@@_put_box_in_flow_bis:nn
2102         \g_@@_left_delim_tl
2103         \g_@@_right_delim_tl
2104     }
2105     \@@_put_box_in_flow:
2106 }

```

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).

```

2107 \bool_if:NT \g_@@_last_col_found_bool
2108 { \skip_horizontal:N \g_@@_width_last_col_dim }
2109 \bool_if:NT \l_@@_preamble_bool
2110 {
2111     \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2112     { \@@_warning_gredirect_none:n { columns~not~used } }
2113 }
2114 \@@_after_array:

```

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

```

2115 \egroup

```

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

```

2116 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2117 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2118 \iow_now:Nx \@mainaux
2119 {
2120     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2121     { \exp_not:o \g_@@_aux_tl }
2122 }
2123 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2124 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2125 }

```

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

## 12 We construct the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble that 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.

```

2126 \cs_new_protected:Npn \@@_transform_preamble:
2127 {
2128     \@@_transform_preamble_i:
2129     \@@_transform_preamble_ii:
2130 }

2131 \cs_new_protected:Npn \@@_transform_preamble_i:
2132 {
2133     \int_gzero:N \c@jCol

```

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

```

2134 \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.

```

2135 \bool_gset_false:N \g_tmpb_bool

```

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

```
2136 \tl_gclear_new:N \g_@@_pre_cell_tl
```

The counter \l\_tmpa\_int will count the number of consecutive occurrences of the symbol |.

```
2137 \int_zero:N \l_tmpa_int
2138 \tl_gclear:N \g_@@_array_preamble_tl
2139 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2140 {
2141   \tl_gset:Nn \g_@@_array_preamble_tl
2142     { ! { \skip_horizontal:N \arrayrulewidth } }
2143 }
2144 {
2145   \clist_if_in:NnT \l_@@_vlines_clist 1
2146   {
2147     \tl_gset:Nn \g_@@_array_preamble_tl
2148       { ! { \skip_horizontal:N \arrayrulewidth } }
2149   }
2150 }
```

Now, we actually make the preamble (which will be given to {array}). It will be stored in \g\_@@\_array\_preamble\_tl.

```
2151 \exp_last_unbraced:N \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2152 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2153 \@@_replace_columncolor:
2154 }
```

```
2155 \hook_gput_code:nnn { begindocument } { . }
2156 {
2157   \IfPackageLoadedTF { colortbl }
2158   {
2159     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2160     \cs_new_protected:Npn \@@_replace_columncolor:
2161     {
2162       \regex_replace_all:NnN
2163         \c_@@_columncolor_regex
2164         { \c { @@_columncolor_preamble } }
2165         \g_@@_array_preamble_tl
2166     }
2167   }
2168   {
2169     \cs_new_protected:Npn \@@_replace_columncolor:
2170     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2171   }
2172 }
```

```
2173 \cs_new_protected:Npn \@@_transform_preamble_ii:
2174 {
```

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

```
2175 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2176 {
2177   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2178   { \bool_gset_true:N \g_@@_delims_bool }
2179 }
2180 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2181 \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).

```

2182 \int_if_zero:nTF \l_@@_first_col_int
2183 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2184 {
2185   \bool_if:NF \g_@@_delims_bool
2186   {
2187     \bool_if:NF \l_@@_tabular_bool
2188     {
2189       \tl_if_empty:NT \l_@@_vlines_clist
2190       {
2191         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2192         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2193       }
2194     }
2195   }
2196 }
2197 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2198 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2199 {
2200   \bool_if:NF \g_@@_delims_bool
2201   {
2202     \bool_if:NF \l_@@_tabular_bool
2203     {
2204       \tl_if_empty:NT \l_@@_vlines_clist
2205       {
2206         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2207         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2208       }
2209     }
2210   }
2211 }

```

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`).

```

2212 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2213 {
2214   \tl_gput_right:Nn \g_@@_array_preamble_tl
2215   { > { \@@_error_too_much_cols: } 1 }
2216 }
2217 }

```

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.

```

2218 \cs_new_protected:Npn \@@_rec_preamble:n #1
2219 {

```

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>

```

2220 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2221 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2222 {

```

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

```

2223 \cs_if_exist:cTF { NC @ find @ #1 }
2224 {
2225   \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }

```

---

<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`.

```

2226         \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2227     }
2228     {
2229         \tl_if_eq:nnT { #1 } { S }
2230         { \@@_fatal:n { unknown~column~type~S } }
2231         { \@@_fatal:nn { unknown~column~type } { #1 } }
2232     }
2233 }
2234 }

```

For c, l and r

```

2235 \cs_new:Npn \@@_c #1
2236 {
2237     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2238     \tl_gclear:N \g_@@_pre_cell_tl
2239     \tl_gput_right:Nn \g_@@_array_preamble_tl
2240     { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2241     \int_gincr:N \c@jCol
2242     \@@_rec_preamble_after_col:n
2243 }

2244 \cs_new:Npn \@@_l #1
2245 {
2246     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2247     \tl_gclear:N \g_@@_pre_cell_tl
2248     \tl_gput_right:Nn \g_@@_array_preamble_tl
2249     {
2250         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2251         l
2252         < \@@_cell_end:
2253     }
2254     \int_gincr:N \c@jCol
2255     \@@_rec_preamble_after_col:n
2256 }

2257 \cs_new:Npn \@@_r #1
2258 {
2259     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2260     \tl_gclear:N \g_@@_pre_cell_tl
2261     \tl_gput_right:Nn \g_@@_array_preamble_tl
2262     {
2263         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2264         r
2265         < \@@_cell_end:
2266     }
2267     \int_gincr:N \c@jCol
2268     \@@_rec_preamble_after_col:n
2269 }

```

For ! and @

```

2270 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2271 {
2272     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2273     \@@_rec_preamble:n
2274 }
2275 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2276 \cs_new:cpn { @@ _ | } #1
2277 {

```

`\l_tmpa_int` is the number of successive occurrences of |

```

2278     \int_incr:N \l_tmpa_int
2279     \@@_make_preamble_i_i:n
2280   }

2281 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2282 {
2283   \str_if_eq:nnTF { #1 } |
2284     { \use:c { @@ _ | } | }
2285     { \@@_make_preamble_i_ii:nn { } #1 }
2286 }

2287 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2288 {
2289   \str_if_eq:nnTF { #2 } [
2290     { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2291     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2292   ]
2293 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2294 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2295 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2296 {
2297   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2298   \tl_gput_right:Nx \g_@@_array_preamble_tl
2299   {

```

Here, the command `\dim_eval:n` is mandatory.

```

2300     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2301   }
2302 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2303 {
2304   \@@_vline:n
2305   {
2306     position = \int_eval:n { \c@jCol + 1 } ,
2307     multiplicity = \int_use:N \l_tmpa_int ,
2308     total-width = \dim_use:N \l_@@_rule_width_dim ,
2309     #2
2310   }

```

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.

```

2311   }
2312   \int_zero:N \l_tmpa_int
2313   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2314   \@@_rec_preamble:n #1
2315 }

2316 \cs_new:cpn { @@ _ > } #1 #2
2317 {
2318   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2319   \@@_rec_preamble:n
2320 }

2321 \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.

```

2322 \keys_define:nn { WithArrows / p-column }
2323 {
2324   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2325   r .value_forbidden:n = true ,
2326   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2327   c .value_forbidden:n = true ,

```

```

2328 l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2329 l .value_forbidden:n = true ,
2330 R .code:n =
2331   \IfPackageLoadedTF { ragged2e }
2332   { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2333   {
2334     \@@_error_or_warning:n { ragged2e~not~loaded }
2335     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2336   } ,
2337 R .value_forbidden:n = true ,
2338 L .code:n =
2339   \IfPackageLoadedTF { ragged2e }
2340   { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2341   {
2342     \@@_error_or_warning:n { ragged2e~not~loaded }
2343     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2344   } ,
2345 L .value_forbidden:n = true ,
2346 C .code:n =
2347   \IfPackageLoadedTF { ragged2e }
2348   { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2349   {
2350     \@@_error_or_warning:n { ragged2e~not~loaded }
2351     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2352   } ,
2353 C .value_forbidden:n = true ,
2354 S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2355 S .value_forbidden:n = true ,
2356 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2357 p .value_forbidden:n = true ,
2358 t .meta:n = p ,
2359 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2360 m .value_forbidden:n = true ,
2361 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2362 b .value_forbidden:n = true ,
2363 }

```

For p, b and m.

```

2364 \cs_new:Npn \@@_p #1
2365 {
2366   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2367   \@@_make_preamble_ii_i:n
2368 }
2369 \cs_set_eq:NN \@@_b \@@_p
2370 \cs_set_eq:NN \@@_m \@@_p

```

```

2371 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2372 {
2373   \str_if_eq:nnTF { #1 } { [ ]
2374     { \@@_make_preamble_ii_ii:w [ ]
2375       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2376     }

```

```

2377 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2378 { \@@_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.

```

2379 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2380 {

```

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).

```

2381 \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2382 % \tl_set:Nn \l_tmpa_tl { #1 }
2383 \@@_keys_p_column:n { #1 }
2384 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2385 }

2386 \cs_new_protected:Npn \@@_keys_p_column:n #1
2387 { \keys_set_known:nnN { WithArrows / 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.

```

2388 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2389 {
2390   \use:e
2391   {
2392     \@@_make_preamble_ii_v:nnnnnnnn
2393     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2394     { \dim_eval:n { #1 } }
2395     {

```

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.

```

2396 \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2397 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2398 {
2399   \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2400   { \str_lowercase:V \l_@@_hpos_col_str }
2401 }
2402 \str_case:on \l_@@_hpos_col_str
2403 {
2404   c { \exp_not:N \centering }
2405   l { \exp_not:N \raggedright }
2406   r { \exp_not:N \raggedleft }
2407   C { \exp_not:N \Centering }
2408   L { \exp_not:N \RaggedRight }
2409   R { \exp_not:N \RaggedLeft }
2410 }
2411 #3
2412 }
2413 { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2414 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2415 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2416 { #2 }
2417 {
2418   \str_case:onF \l_@@_hpos_col_str
2419   {
2420     { j } { c }
2421     { si } { c }
2422   }

```

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

```

2423 { \str_lowercase:V \l_@@_hpos_col_str }
2424 }
2425 }

```

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

```

2426 \int_gincr:N \c_jCol
2427 \@@_rec_preamble_after_col:n
2428 }

```

#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*.

```

2429 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2430 {
2431   \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2432     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2433     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2434   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2435   \tl_gclear:N \g_@@_pre_cell_tl
2436   \tl_gput_right:Nn \g_@@_array_preamble_tl
2437     {
2438       > {

```

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.

```

2439       \dim_set:Nn \l_@@_col_width_dim { #2 }
2440       \@@_cell_begin:w

```

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

```

2441       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2442   \everypar
2443   {
2444     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2445     \everypar { }
2446   }

```

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

```

2447   #3

```

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

```

2448   \g_@@_row_style_tl
2449   \arraybackslash
2450   #5
2451   }
2452   #8
2453   < {
2454     #6

```

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

```

2455   \@finalstrut \@arstrutbox
2456   \use:c { end #7 }

```

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

```

2457   #4
2458   \@@_cell_end:
2459   }
2460 }
2461 }

```

```

2462 \str_new:N \c_@@_ignorespaces_str
2463 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2464 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }

```

In order to test whether a cell is empty, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstances, for example when `\collectcell` of `collcell` is used, the cell does not begin with `\ignorespaces`. In that case, we consider as not empty...

First, we test if the next token is `\ignorespaces` and it's not very easy...

```

2465 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2466 \cs_new_protected:Npn \@@_test_if_empty_i:
2467 {
2468   \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2469   \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2470   { \@@_test_if_empty:w }
2471 }
2472 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2473 {
2474   \peek_meaning:NT \unskip
2475   {
2476     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2477     {
2478       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2479       \skip_horizontal:N \l_@@_col_width_dim
2480     }
2481   }
2482 }
2483 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2484 {
2485   \peek_meaning:NT \__siunitx_table_skip:n
2486   {
2487     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2488     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2489   }
2490 }

```

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 `\@arstrutbox`, there is only one row.

```

2491 \cs_new_protected:Npn \@@_center_cell_box:
2492 {

```

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

```

2493   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2494   {
2495     \int_compare:nNnT
2496     { \box_ht:N \l_@@_cell_box }
2497     >

```

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).

```

2498     { \box_ht:N \strutbox }
2499     {
2500       \hbox_set:Nn \l_@@_cell_box
2501       {
2502         \box_move_down:nn
2503         {
2504           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2505             + \baselineskip ) / 2
2506         }
2507         { \box_use:N \l_@@_cell_box }
2508     }
2509 }
2510 }
2511 }

```

For V (similar to the V of varwidth).

```

2512 \cs_new:Npn \@@_V #1 #2
2513 {
2514     \str_if_eq:nnTF { #2 } { [ ] }
2515     { \@@_make_preamble_V_i:w [ ] }
2516     { \@@_make_preamble_V_i:w [ ] { #2 } }
2517 }
2518 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2519 { \@@_make_preamble_V_ii:nn { #1 } }
2520 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2521 {
2522     \str_set:Nn \l_@@_vpos_col_str { p }
2523     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2524     \@@_keys_p_column:n { #1 }
2525     \IfPackageLoadedTF { varwidth }
2526     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2527     {
2528         \@@_error_or_warning:n { varwidth-not-loaded }
2529         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2530     }
2531 }

```

For w and W

```

2532 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2533 \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.

```

2534 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2535 {
2536     \str_if_eq:nnTF { #3 } { s }
2537     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2538     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2539 }

```

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.

```

2540 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2541 {
2542     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2543     \tl_gclear:N \g_@@_pre_cell_tl
2544     \tl_gput_right:Nn \g_@@_array_preamble_tl
2545     {
2546         > {
2547             \dim_set:Nn \l_@@_col_width_dim { #2 }
2548             \@@_cell_begin:w
2549             \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2550         }
2551         c
2552         < {
2553             \@@_cell_end_for_w_s:

```

```

2554         #1
2555         \@@_adjust_size_box:
2556         \box_use_drop:N \l_@@_cell_box
2557     }
2558 }
2559 \int_gincr:N \c@jCol
2560 \@@_rec_preamble_after_col:n
2561 }

```

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

```

2562 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2563 {
2564     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2565     \tl_gclear:N \g_@@_pre_cell_tl
2566     \tl_gput_right:Nn \g_@@_array_preamble_tl
2567     {
2568         > {

```

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.

```

2569         \dim_set:Nn \l_@@_col_width_dim { #4 }
2570         \hbox_set:Nw \l_@@_cell_box
2571         \@@_cell_begin:w
2572         \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2573     }
2574     c
2575     < {
2576         \@@_cell_end:
2577         \hbox_set_end:
2578         #1
2579         \@@_adjust_size_box:
2580         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2581     }
2582 }

```

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

```

2583     \int_gincr:N \c@jCol
2584     \@@_rec_preamble_after_col:n
2585 }

```

```

2586 \cs_new_protected:Npn \@@_special_W:
2587 {
2588     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2589     { \@@_warning:n { W~warning } }
2590 }

```

For S (of siunitx).

```

2591 \cs_new:Npn \@@_S #1 #2
2592 {
2593     \str_if_eq:nnTF { #2 } { [ ] }
2594     { \@@_make_preamble_S:w [ ] }
2595     { \@@_make_preamble_S:w [ ] { #2 } }
2596 }
2597 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2598 { \@@_make_preamble_S:i:n { #1 } }
2599 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2600 {
2601     \IfPackageLoadedTF { siunitx }
2602     {
2603         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2604         \tl_gclear:N \g_@@_pre_cell_tl
2605         \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2606     {
2607     > {
2608         \@@_cell_begin:w
2609         \keys_set:nn { siunitx } { #1 }
2610         \siunitx_cell_begin:w
2611     }
2612     c
2613     < { \siunitx_cell_end: \@@_cell_end: }
2614 }

```

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

```

2615     \int_gincr:N \c@jCol
2616     \@@_rec_preamble_after_col:n
2617 }
2618 { \@@_fatal:n { siunitx~not~loaded } }
2619 }

```

For (, [ and \{.

```

2620 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2621 {
2622     \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.

```

2623     \int_if_zero:nTF \c@jCol
2624     {
2625         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2626         {

```

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

```

2627         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2628         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2629         \@@_rec_preamble:n #2
2630     }
2631     {
2632         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2633         \@@_make_preamble_iv:nn { #1 } { #2 }
2634     }
2635 }
2636 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2637 }
2638 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2639 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2640 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2641 {
2642     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2643     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2644     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2645     {
2646         \@@_error:nn { delimiter~after~opening } { #2 }
2647         \@@_rec_preamble:n
2648     }
2649     { \@@_rec_preamble:n #2 }
2650 }

```

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

```

2651 \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}).

```

2652 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2

```

```

2653 {
2654   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2655   \tl_if_in:nnTF { } ] \} { #2 }
2656   { \@@_make_preamble_v:nnn #1 #2 }
2657   {
2658     \tl_if_eq:nnTF { \stop } { #2 }
2659     {
2660       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2661       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2662       {
2663         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2664         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2665         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2666         \@@_rec_preamble:n #2
2667       }
2668     }
2669     {
2670       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2671       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2672       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2673       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2674       \@@_rec_preamble:n #2
2675     }
2676   }
2677 }
2678 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ] }
2679 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2680 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2681 {
2682   \tl_if_eq:nnTF { \stop } { #3 }
2683   {
2684     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2685     {
2686       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2687       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2688       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2689       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2690     }
2691     {
2692       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2693       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2694       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2695       \@@_error:nn { double~closing~delimiter } { #2 }
2696     }
2697   }
2698   {
2699     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2700     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2701     \@@_error:nn { double~closing~delimiter } { #2 }
2702     \@@_rec_preamble:n #3
2703   }
2704 }
2705 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2706 { \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 @{...}.

```

2707 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2708 {
2709   \str_if_eq:nnTF { #1 } { < }

```

```

2710 \@@_rec_preamble_after_col_i:n
2711 {
2712   \str_if_eq:nnTF { #1 } { @ }
2713   \@@_rec_preamble_after_col_ii:n
2714   {
2715     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2716     {
2717       \tl_gput_right:Nn \g_@@_array_preamble_tl
2718       { ! { \skip_horizontal:N \arrayrulewidth } }
2719     }
2720     {
2721       \exp_args:NNe
2722       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2723       {
2724         \tl_gput_right:Nn \g_@@_array_preamble_tl
2725         { ! { \skip_horizontal:N \arrayrulewidth } }
2726       }
2727     }
2728     \@@_rec_preamble:n { #1 }
2729   }
2730 }
2731 }
2732 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2733 {
2734   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2735   \@@_rec_preamble_after_col:n
2736 }

```

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.

```

2737 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2738 {
2739   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2740   {
2741     \tl_gput_right:Nn \g_@@_array_preamble_tl
2742     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2743   }
2744   {
2745     \exp_args:NNe
2746     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2747     {
2748       \tl_gput_right:Nn \g_@@_array_preamble_tl
2749       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2750     }
2751     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2752   }
2753   \@@_rec_preamble:n
2754 }

```

```

2755 \cs_new:cpn { @@ _ * } #1 #2 #3
2756 {
2757   \tl_clear:N \l_tmpa_tl
2758   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2759   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2760 }

```

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

```

2761 \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.

```

2762 \cs_new:Npn \@@_X #1 #2
2763 {
2764   \str_if_eq:nnTF { #2 } { [ ]
2765     { \@@_make_preamble_X:w [ ]
2766       { \@@_make_preamble_X:w [ ] #2 }
2767     }
2768   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2769     { \@@_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 { WithArrows / 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).

```

2770 \keys_define:nn { WithArrows / X-column }
2771 { 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.

```

2772 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2773 {

```

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).

```

2774   \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).

```

2775   \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.

```

2776   \int_zero_new:N \l_@@_weight_int
2777   \int_set_eq:NN \l_@@_weight_int \c_one_int
2778   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in \l\_tmpa\_tl

```

2779   \keys_set:no { WithArrows / X-column } \l_tmpa_tl
2780   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2781   {
2782     \@@_error_or_warning:n { negative-weight }
2783     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2784   }
2785   \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).

```

2786   \bool_if:NTF \l_@@_X_columns_aux_bool
2787   {
2788     \exp_args:Nne
2789     \@@_make_preamble_ii_iv:nnn
2790     { \l_@@_weight_int \l_@@_X_columns_dim }
2791     { minipage }
2792     { \@@_no_update_width: }
2793   }
2794   {
2795     \tl_gput_right:Nn \g_@@_array_preamble_tl
2796     {
2797       > {
2798         \@@_cell_begin:w
2799         \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`.

```
2800 \NotEmpty
```

The following code will nullify the box of the cell.

```
2801 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2802 { \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`.

```
2803 \begin { minipage } { 5 cm } \arraybackslash
2804 }
2805 c
2806 < {
2807 \end { minipage }
2808 \@@_cell_end:
2809 }
2810 }
2811 \int_gincr:N \c@jCol
2812 \@@_rec_preamble_after_col:n
2813 }
2814 }

2815 \cs_new_protected:Npn \@@_no_update_width:
2816 {
2817 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2818 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2819 }
```

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

```
2820 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2821 {
2822 \seq_gput_right:Nx \g_@@_cols_vlism_seq
2823 { \int_eval:n { \c@jCol + 1 } }
2824 \tl_gput_right:Nx \g_@@_array_preamble_tl
2825 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2826 \@@_rec_preamble:n
2827 }
```

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.

```
2828 \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).

```
2829 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2830 { \@@_fatal:n { Preamble~forgotten } }
2831 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2832 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2833 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
```

## 13 The redefinition of `\multicolumn`

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

```
2834 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2835 {
```

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 that one column specifier in the preamble of `\multicolumn`.

```

2836 \multispan { #1 }
2837 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2838 \begingroup
2839 \cs_set:Npn \@addamp
2840 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2841 \tl_gclear:N \g_@@_preamble_tl
2842 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2843 \exp_args:No \@mkpream \g_@@_preamble_tl
2844 \@addtopreamble \empty
2845 \endgroup

```

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

```

2846 \int_compare:nNnT { #1 } > \c_one_int
2847 {
2848   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2849   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2850   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2851   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2852   {
2853     {
2854       \int_if_zero:nTF \c@jCol
2855       { \int_eval:n { \c@iRow + 1 } }
2856       { \int_use:N \c@iRow }
2857     }
2858     { \int_eval:n { \c@jCol + 1 } }
2859     {
2860       \int_if_zero:nTF \c@jCol
2861       { \int_eval:n { \c@iRow + 1 } }
2862       { \int_use:N \c@iRow }
2863     }
2864     { \int_eval:n { \c@jCol + #1 } }
2865     { } % for the name of the block
2866   }
2867 }

```

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

```

2868 \cs_set:Npn \@sharp { #3 }
2869 \@arstrut
2870 \@preamble
2871 \null

```

We add some lines.

```

2872 \int_gadd:Nn \c@jCol { #1 - 1 }
2873 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2874 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2875 \ignorespaces
2876 }

```

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`.

```

2877 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2878 {
2879   \str_case:nnF { #1 }
2880   {

```

```

2881 c { \@@_make_m_preamble_i:n #1 }
2882 l { \@@_make_m_preamble_i:n #1 }
2883 r { \@@_make_m_preamble_i:n #1 }
2884 > { \@@_make_m_preamble_ii:nn #1 }
2885 ! { \@@_make_m_preamble_ii:nn #1 }
2886 @ { \@@_make_m_preamble_ii:nn #1 }
2887 | { \@@_make_m_preamble_iii:n #1 }
2888 p { \@@_make_m_preamble_iv:nnn t #1 }
2889 m { \@@_make_m_preamble_iv:nnn c #1 }
2890 b { \@@_make_m_preamble_iv:nnn b #1 }
2891 w { \@@_make_m_preamble_v:nnnn { } #1 }
2892 W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2893 \q_stop { }
2894 }
2895 {
2896 \cs_if_exist:cTF { NC @ find @ #1 }
2897 {
2898 \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2899 \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2900 }
2901 {
2902 \tl_if_eq:nnT { #1 } { S }
2903 { \@@_fatal:n { unknown~column~type~S } }
2904 { \@@_fatal:nn { unknown~column~type } { #1 } }
2905 }
2906 }
2907 }

```

For c, l and r

```

2908 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2909 {
2910 \tl_gput_right:Nn \g_@@_preamble_tl
2911 {
2912 > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2913 #1
2914 < \@@_cell_end:
2915 }

```

We test for the presence of a <.

```

2916 \@@_make_m_preamble_x:n
2917 }

```

For >, ! and @

```

2918 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2919 {
2920 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2921 \@@_make_m_preamble:n
2922 }

```

For |

```

2923 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2924 {
2925 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2926 \@@_make_m_preamble:n
2927 }

```

For p, m and b

```

2928 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2929 {
2930 \tl_gput_right:Nn \g_@@_preamble_tl
2931 {
2932 > {
2933 \@@_cell_begin:w

```

```

2934         \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2935         \mode_leave_vertical:
2936         \arraybackslash
2937         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2938     }
2939     c
2940     < {
2941         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2942         \end { minipage }
2943         \@@_cell_end:
2944     }
2945 }

```

We test for the presence of a <.

```

2946     \@@_make_m_preamble_x:n
2947 }

```

For w and W

```

2948 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2949 {
2950     \tl_gput_right:Nn \g_@@_preamble_tl
2951     {
2952         > {
2953             \dim_set:Nn \l_@@_col_width_dim { #4 }
2954             \hbox_set:Nw \l_@@_cell_box
2955             \@@_cell_begin:w
2956             \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2957         }
2958         c
2959         < {
2960             \@@_cell_end:
2961             \hbox_set_end:
2962             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2963             #1
2964             \@@_adjust_size_box:
2965             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2966         }
2967     }
}

```

We test for the presence of a <.

```

2968     \@@_make_m_preamble_x:n
2969 }

```

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

```

2970 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2971 {
2972     \str_if_eq:nnTF { #1 } { < }
2973     \@@_make_m_preamble_ix:n
2974     { \@@_make_m_preamble:n { #1 } }
2975 }
2976 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2977 {
2978     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2979     \@@_make_m_preamble_x:n
2980 }

```

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).

```

2981 \cs_new_protected:Npn \@@_put_box_in_flow:
2982 {
2983     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }

```

```

2984 \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2985 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2986 { \box_use_drop:N \l_tmpa_box }
2987 \@@_put_box_in_flow_i:
2988 }

```

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).

```

2989 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2990 {
2991   \pgfpicture
2992     \@@_qpoint:n { row - 1 }
2993     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2994     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2995     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2996     \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).

```

2997 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2998 {
2999   \int_set:Nn \l_tmpa_int
3000   {
3001     \str_range:Nnn
3002       \l_@@_baseline_tl
3003       6
3004       { \tl_count:o \l_@@_baseline_tl }
3005   }
3006   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3007 }
3008 {
3009   \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3010   { \int_set_eq:NN \l_tmpa_int \c_one_int }
3011   {
3012     \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3013     { \int_set_eq:NN \l_tmpa_int \c@iRow }
3014     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3015   }
3016   \bool_lazy_or:nnT
3017   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3018   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3019   {
3020     \@@_error:n { bad-value-for-baseline }
3021     \int_set_eq:NN \l_tmpa_int \c_one_int
3022   }
3023   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3024 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3025 }
3026 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3027 \endpgfpicture
3028 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3029 \box_use_drop:N \l_tmpa_box
3030 }

```

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).

```

3031 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3032 {

```

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.

```

3033 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3034 {
3035   \int_compare:nNnT \c@jCol > \c_one_int
3036   {
3037     \box_set_wd:Nn \l_@@_the_array_box
3038     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3039   }
3040 }

```

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

```

3041 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3042 \bool_if:NT \l_@@_caption_above_bool
3043 {
3044   \tl_if_empty:NF \l_@@_caption_tl
3045   {
3046     \bool_set_false:N \g_@@_caption_finished_bool
3047     \int_gzero:N \c@tabularnote
3048     \@@_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).

```

3049 \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3050 {
3051   \tl_gput_right:Nx \g_@@_aux_tl
3052   {
3053     \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3054     { \int_use:N \g_@@_notes_caption_int }
3055   }
3056   \int_gzero:N \g_@@_notes_caption_int
3057 }
3058 }
3059 }

```

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

```

3060 \hbox
3061 {
3062   \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.

```

3063 \@@_create_extra_nodes:
3064 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3065 }

```

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).

```

3066 \bool_lazy_any:nT
3067 {
3068   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3069   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3070   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3071 }
3072 \@@_insert_tabularnotes:
3073 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3074 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:

```

```

3075 \end { minipage }
3076 }

```

```

3077 \cs_new_protected:Npn \@@_insert_caption:
3078 {
3079   \tl_if_empty:NF \l_@@_caption_tl
3080   {
3081     \cs_if_exist:NTF \@capttype
3082     { \@@_insert_caption_i: }
3083     { \@@_error:n { caption~outside~float } }
3084   }
3085 }

```

```

3086 \cs_new_protected:Npn \@@_insert_caption_i:
3087 {
3088   \group_begin:

```

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

```

3089   \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.

```

3090   \IfPackageLoadedTF { floatrow }
3091   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3092   { }
3093   \tl_if_empty:NTF \l_@@_short_caption_tl
3094   { \caption }
3095   { \caption [ \l_@@_short_caption_tl ] }
3096   { \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.

```

3097   \bool_if:NF \g_@@_caption_finished_bool
3098   {
3099     \bool_gset_true:N \g_@@_caption_finished_bool
3100     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3101     \int_gzero:N \c@tabularnote
3102   }
3103   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3104   \group_end:
3105 }

```

```

3106 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3107 {
3108   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3109   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3110 }

```

```

3111 \cs_new_protected:Npn \@@_insert_tabularnotes:
3112 {
3113   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3114   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3115   \skip_vertical:N 0.65ex

```

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

```

3116   \group_begin:
3117   \l_@@_notes_code_before_tl
3118   \tl_if_empty:NF \g_@@_tabularnote_tl
3119   {

```

```

3120     \g_@@_tabularnote_tl \par
3121     \tl_gclear:N \g_@@_tabularnote_tl
3122 }

```

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.

```

3123     \int_compare:nNnT \c@tabularnote > \c_zero_int
3124     {
3125         \bool_if:NTF \l_@@_notes_para_bool
3126         {
3127             \begin { tabularnotes* }
3128             \seq_map_inline:Nn \g_@@_notes_seq
3129             { \@@_one_tabularnote:nn ##1 }
3130             \strut
3131             \end { tabularnotes* }

```

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

```

3132     \par
3133 }
3134 {
3135     \tabularnotes
3136     \seq_map_inline:Nn \g_@@_notes_seq
3137     { \@@_one_tabularnote:nn ##1 }
3138     \strut
3139     \endtabularnotes
3140 }
3141 }
3142 \unskip
3143 \group_end:
3144 \bool_if:NT \l_@@_notes_bottomrule_bool
3145 {
3146     \IfPackageLoadedTF { booktabs }
3147     {

```

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

```

3148     \skip_vertical:N \aboverulesep

```

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

```

3149     { \CT@arc@ \hrule height \heavyrulewidth }
3150 }
3151 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3152 }
3153 \l_@@_notes_code_after_tl
3154 \seq_gclear:N \g_@@_notes_seq
3155 \seq_gclear:N \g_@@_notes_in_caption_seq
3156 \int_gzero:N \c@tabularnote
3157 }

```

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.

```

3158 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3159 {
3160     \tl_if_novalue:nTF { #1 }
3161     { \item }
3162     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3163 }

```

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`.

```

3164 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:

```

```

3165 {
3166   \pgfpicture
3167     \@@_qpoint:n { row - 1 }
3168     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3169     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3170     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3171   \endpgfpicture
3172   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3173   \int_if_zero:nT \l_@@_first_row_int
3174   {
3175     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3176     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3177   }
3178   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3179 }

```

Now, the general case.

```

3180 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3181 {

```

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

```

3182   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3183   { \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`.

```

3184   \pgfpicture
3185   \@@_qpoint:n { row - 1 }
3186   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3187   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3188   {
3189     \int_set:Nn \l_tmpa_int
3190     {
3191       \str_range:Nnn
3192         \l_@@_baseline_tl
3193         6
3194         { \tl_count:o \l_@@_baseline_tl }
3195     }
3196     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3197   }
3198   {
3199     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3200     \bool_lazy_or:nnT
3201       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3202       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3203     {
3204       \@@_error:n { bad~value~for~baseline }
3205       \int_set:Nn \l_tmpa_int 1
3206     }
3207     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3208   }
3209   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3210   \endpgfpicture
3211   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3212   \int_if_zero:nT \l_@@_first_row_int
3213   {
3214     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3215     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3216   }
3217   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3218 }

```

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.

```
3219 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3220 {
```

We will compute the real width of both delimiters used.

```
3221 \dim_zero_new:N \l_@@_real_left_delim_dim
3222 \dim_zero_new:N \l_@@_real_right_delim_dim
3223 \hbox_set:Nn \l_tmpb_box
3224 {
3225   \c_math_toggle_token
3226   \left #1
3227   \vcenter
3228   {
3229     \vbox_to_ht:nn
3230     { \box_ht_plus_dp:N \l_tmpa_box }
3231     { }
3232   }
3233   \right .
3234   \c_math_toggle_token
3235 }
3236 \dim_set:Nn \l_@@_real_left_delim_dim
3237 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3238 \hbox_set:Nn \l_tmpb_box
3239 {
3240   \c_math_toggle_token
3241   \left .
3242   \vbox_to_ht:nn
3243   { \box_ht_plus_dp:N \l_tmpa_box }
3244   { }
3245   \right #2
3246   \c_math_toggle_token
3247 }
3248 \dim_set:Nn \l_@@_real_right_delim_dim
3249 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3250 \skip_horizontal:N \l_@@_left_delim_dim
3251 \skip_horizontal:N -\l_@@_real_left_delim_dim
3252 \@@_put_box_in_flow:
3253 \skip_horizontal:N \l_@@_right_delim_dim
3254 \skip_horizontal:N -\l_@@_real_right_delim_dim
3255 }
```

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).

```
3256 \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).

```
3257 {
3258   \peek_remove_spaces:n
3259   {
3260     \peek_meaning:NTF \end
3261     \@@_analyze_end:Nn
3262     {
3263       \@@_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`).

```

3264         \exp_args:No \@@_array: \g_@@_array_preamble_tl
3265     }
3266 }
3267 }
3268 {
3269     \@@_create_col_nodes:
3270     \endarray
3271 }

```

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

```

3272 \NewDocumentEnvironment { @@-light-syntax } { b }
3273 {

```

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`.

```

3274     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3275     \tl_map_inline:nn { #1 }
3276     {
3277         \str_if_eq:nnT { ##1 } { & }
3278         { \@@_fatal:n { ampersand-in~light-syntax } }
3279         \str_if_eq:nnT { ##1 } { \ }
3280         { \@@_fatal:n { double-backslash-in~light-syntax } }
3281     }

```

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`.

```

3282     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3283 }

```

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.

```

3284 {
3285     \@@_create_col_nodes:
3286     \endarray
3287 }
3288 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3289 {
3290     \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).

```

3291     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3292     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3293     \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3294     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3295     \tl_if_empty:NF \l_tmpa_tl
3296     { \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.

```

3297   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3298   { \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`).

```

3299   \tl_build_begin:N \l_@@_new_body_tl
3300   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3301   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3302   \@@_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).

```

3303   \seq_map_inline:Nn \l_@@_rows_seq
3304   {
3305     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3306     \@@_line_with_light_syntax:n { #1 }
3307   }
3308   \tl_build_end:N \l_@@_new_body_tl
3309   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3310   {
3311     \int_set:Nn \l_@@_last_col_int
3312     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3313   }

```

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.

```

3314   \@@_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`).

```

3315   \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3316   }
3317   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3318   {
3319     \seq_clear_new:N \l_@@_cells_seq
3320     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3321     \int_set:Nn \l_@@_nb_cols_int
3322     {
3323       \int_max:nn
3324       \l_@@_nb_cols_int
3325       { \seq_count:N \l_@@_cells_seq }
3326     }
3327     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3328     \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3329     \seq_map_inline:Nn \l_@@_cells_seq
3330     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3331   }
3332   \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }

```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security). When this command is used, `#1` is, in fact, always `\end`.

```

3333   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3334   {
3335     \str_if_eq:onT \g_@@_name_env_str { #2 }
3336     { \@@_fatal:n { empty-environment } }

```

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

```
3337   \end { #2 }
3338 }
```

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).

```
3339 \cs_new:Npn \@@_create_col_nodes:
3340 {
3341   \crr
3342   \int_if_zero:nT \l_@@_first_col_int
3343   {
3344     \omit
3345     \hbox_overlap_left:n
3346     {
3347       \bool_if:NT \l_@@_code_before_bool
3348       { \pgfsys@markposition { \@@_env: - col - 0 } }
3349       \pgfpicture
3350       \pgfrememberpicturepositiononpagetrue
3351       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3352       \str_if_empty:NF \l_@@_name_str
3353       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3354       \endpgfpicture
3355       \skip_horizontal:N 2\col@sep
3356       \skip_horizontal:N \g_@@_width_first_col_dim
3357     }
3358     &
3359   }
3360   \omit
```

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

```
3361   \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`).

```
3362   \int_if_zero:nTF \l_@@_first_col_int
3363   {
3364     \bool_if:NT \l_@@_code_before_bool
3365     {
3366       \hbox
3367       {
3368         \skip_horizontal:N -0.5\arrayrulewidth
3369         \pgfsys@markposition { \@@_env: - col - 1 }
3370         \skip_horizontal:N 0.5\arrayrulewidth
3371       }
3372     }
3373     \pgfpicture
3374     \pgfrememberpicturepositiononpagetrue
3375     \pgfcoordinate { \@@_env: - col - 1 }
3376     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3377     \str_if_empty:NF \l_@@_name_str
3378     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3379     \endpgfpicture
3380   }
3381   {
3382     \bool_if:NT \l_@@_code_before_bool
3383     {
3384       \hbox
3385       {
3386         \skip_horizontal:N 0.5\arrayrulewidth
3387         \pgfsys@markposition { \@@_env: - col - 1 }
3388         \skip_horizontal:N -0.5\arrayrulewidth
```

```

3389     }
3390   }
3391   \pgfpicture
3392   \pgfrememberpicturepositiononpagetrue
3393   \pgfcoordinate { \@@_env: - col - 1 }
3394   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3395   \str_if_empty:NF \l_@@_name_str
3396   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3397   \endpgfpicture
3398 }

```

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.

```

3399   \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3400   \bool_if:NF \l_@@_auto_columns_width_bool
3401   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3402   {
3403     \bool_lazy_and:nnTF
3404       \l_@@_auto_columns_width_bool
3405       { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3406       { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3407       { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3408     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3409   }
3410   \skip_horizontal:N \g_tmpa_skip
3411   \hbox
3412   {
3413     \bool_if:NT \l_@@_code_before_bool
3414     {
3415       \hbox
3416       {
3417         \skip_horizontal:N -0.5\arrayrulewidth
3418         \pgfsys@markposition { \@@_env: - col - 2 }
3419         \skip_horizontal:N 0.5\arrayrulewidth
3420       }
3421     }
3422     \pgfpicture
3423     \pgfrememberpicturepositiononpagetrue
3424     \pgfcoordinate { \@@_env: - col - 2 }
3425     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3426     \str_if_empty:NF \l_@@_name_str
3427     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3428     \endpgfpicture
3429   }

```

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.

```

3430   \int_gset_eq:NN \g_tmpa_int \c_one_int
3431   \bool_if:NTF \g_@@_last_col_found_bool
3432   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3433   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3434   {
3435     &
3436     \omit
3437     \int_gincr:N \g_tmpa_int

```

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

```

3438     \skip_horizontal:N \g_tmpa_skip
3439     \bool_if:NT \l_@@_code_before_bool
3440     {

```

```

3441         \hbox
3442         {
3443             \skip_horizontal:N -0.5\arrayrulewidth
3444             \pgfsys@markposition
3445             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3446             \skip_horizontal:N 0.5\arrayrulewidth
3447         }
3448     }

```

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

```

3449     \pgfpicture
3450     \pgfrememberpicturepositiononpagetrue
3451     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3452     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3453     \str_if_empty:NF \l_@@_name_str
3454     {
3455         \pgfnodealias
3456         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3457         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3458     }
3459     \endpgfpicture
3460 }

```

```

3461 &
3462 \omit

```

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

```

3463 \int_if_zero:nT \g_@@_col_total_int
3464 { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3465 \skip_horizontal:N \g_tmpa_skip
3466 \int_gincr:N \g_tmpa_int
3467 \bool_lazy_any:nF % modified 2023/12/13
3468 {
3469     \g_@@_delims_bool
3470     \l_@@_tabular_bool
3471     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3472     \l_@@_exterior_arraycolsep_bool
3473     \l_@@_bar_at_end_of_pream_bool
3474 }
3475 { \skip_horizontal:N -\col@sep }
3476 \bool_if:NT \l_@@_code_before_bool
3477 {
3478     \hbox
3479     {
3480         \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.

```

3481         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3482         { \skip_horizontal:N -\arraycolsep }
3483         \pgfsys@markposition
3484         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3485         \skip_horizontal:N 0.5\arrayrulewidth
3486         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3487         { \skip_horizontal:N \arraycolsep }
3488     }
3489 }
3490 \pgfpicture
3491 \pgfrememberpicturepositiononpagetrue
3492 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3493 {
3494     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3495         {
3496             \pgfpoint
3497             { - 0.5 \arrayrulewidth - \arraycolsep }
3498             \c_zero_dim
3499         }
3500         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3501     }
3502     \str_if_empty:NF \l_@@_name_str
3503     {
3504         \pgfnodealias
3505         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3506         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3507     }
3508     \endpgfpicture

3509     \bool_if:NT \g_@@_last_col_found_bool
3510     {
3511         \hbox_overlap_right:n
3512         {
3513             \skip_horizontal:N \g_@@_width_last_col_dim
3514             \skip_horizontal:N \col@sep % added 2023-11-05
3515             \bool_if:NT \l_@@_code_before_bool
3516             {
3517                 \pgfsys@markposition
3518                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3519             }
3520             \pgfpicture
3521             \pgfrememberpicturepositiononpagetrue
3522             \pgfcoordinate
3523             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3524             \pgfpointorigin
3525             \str_if_empty:NF \l_@@_name_str
3526             {
3527                 \pgfnodealias
3528                 {
3529                     \l_@@_name_str - col
3530                     - \int_eval:n { \g_@@_col_total_int + 1 }
3531                 }
3532                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3533             }
3534             \endpgfpicture
3535         }
3536     }
3537     \cr
3538 }

```

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

```

3539 \tl_const:Nn \c_@@_preamble_first_col_tl
3540 {
3541     >
3542     {

```

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).

```

3543     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3544     \bool_gset_true:N \g_@@_after_col_zero_bool
3545     \@@_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.

```

3546     \hbox_set:Nw \l_@@_cell_box
3547     \@@_math_toggle:
3548     \@@_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”.

```

3549     \int_compare:nNnT \c@iRow > \c_zero_int
3550     {
3551         \bool_lazy_or:nnT
3552         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3553         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3554         {
3555             \l_@@_code_for_first_col_tl
3556             \xglobal \colorlet { nicematrix-first-col } { . }
3557         }
3558     }
3559 }
```

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`.

```

3560     l
3561     <
3562     {
3563         \@@_math_toggle:
3564         \hbox_set_end:
3565         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3566         \@@_adjust_size_box:
3567         \@@_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.

```

3568     \dim_gset:Nn \g_@@_width_first_col_dim
3569     { \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.

```

3570     \hbox_overlap_left:n
3571     {
3572         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3573         \@@_node_for_cell:
3574         { \box_use_drop:N \l_@@_cell_box }
3575         \skip_horizontal:N \l_@@_left_delim_dim
3576         \skip_horizontal:N \l_@@_left_margin_dim
3577         \skip_horizontal:N \l_@@_extra_left_margin_dim
3578     }
3579     \bool_gset_false:N \g_@@_empty_cell_bool
3580     \skip_horizontal:N -2\col@sep
3581 }
3582 }
```

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

```

3583 \tl_const:Nn \c_@@_preamble_last_col_tl
3584 {
3585     >
3586     {
3587         \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).

```

3588     \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.

```

3589     \bool_gset_true:N \g_@@_last_col_found_bool
3590     \int_gincr:N \c@jCol
3591     \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.

```

3592     \hbox_set:Nw \l_@@_cell_box
3593     \@@_math_toggle:
3594     \@@_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”.

```

3595     \int_compare:nNnT \c@iRow > \c_zero_int
3596     {
3597         \bool_lazy_or:nnT
3598         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3599         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3600         {
3601             \l_@@_code_for_last_col_tl
3602             \xglobal \colorlet { nicematrix-last-col } { . }
3603         }
3604     }
3605 }
3606 1
3607 <
3608 {
3609     \@@_math_toggle:
3610     \hbox_set_end:
3611     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3612     \@@_adjust_size_box:
3613     \@@_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.

```

3614     \dim_gset:Nn \g_@@_width_last_col_dim
3615     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3616     \skip_horizontal:N -2\col@sep

```

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

```

3617     \hbox_overlap_right:n
3618     {
3619         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3620         {
3621             \skip_horizontal:N \l_@@_right_delim_dim
3622             \skip_horizontal:N \l_@@_right_margin_dim
3623             \skip_horizontal:N \l_@@_extra_right_margin_dim
3624             \@@_node_for_cell:
3625         }
3626     }
3627     \bool_gset_false:N \g_@@_empty_cell_bool
3628 }
3629 }

```

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

```

3630 \NewDocumentEnvironment { NiceArray } { }
3631 {
3632     \bool_gset_false:N \g_@@_delims_bool
3633     \str_if_empty:NT \g_@@_name_env_str
3634     { \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).

```

3635     \NiceArrayWithDelims . .
3636 }
3637 { \endNiceArrayWithDelims }

```

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

```

3638 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3639 {
3640     \NewDocumentEnvironment { #1 NiceArray } { }
3641     {

```

```

3642     \bool_gset_true:N \g_@@_delims_bool
3643     \str_if_empty:NT \g_@@_name_env_str
3644     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3645     \@@_test_if_math_mode:
3646     \NiceArrayWithDelims #2 #3
3647   }
3648   { \endNiceArrayWithDelims }
3649 }
3650 \@@_def_env:nnn p ( )
3651 \@@_def_env:nnn b [ ]
3652 \@@_def_env:nnn B \{ \}
3653 \@@_def_env:nnn v | |
3654 \@@_def_env:nnn V \l \l

```

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

```

3655 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3656 {
3657   \bool_set_false:N \l_@@_preamble_bool
3658   \tl_clear:N \l_tmpa_tl
3659   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3660   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3661   \tl_put_right:Nn \l_tmpa_tl
3662   {
3663     *
3664     {
3665       \int_case:nnF \l_@@_last_col_int
3666       {
3667         { -2 } { \c@MaxMatrixCols }
3668         { -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).

```

3669       }
3670       { \int_eval:n { \l_@@_last_col_int - 1 } }
3671     }
3672     { #2 }
3673   }
3674   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3675   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3676 }
3677 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3678 \clist_map_inline:nn { p , b , B , v , V }
3679 {
3680   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3681   {
3682     \bool_gset_true:N \g_@@_delims_bool
3683     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3684     % added 2023/10/01
3685     \int_if_zero:nT \l_@@_last_col_int
3686     {
3687       \bool_set_true:N \l_@@_last_col_without_value_bool
3688       \int_set:Nn \l_@@_last_col_int { -1 }
3689     }
3690     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3691     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3692   }
3693   { \use:c { end #1 NiceArray } }
3694 }

```

We define also an environment `{NiceMatrix}`

```

3695 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3696 {
3697   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3698   % added 2023/10/01
3699   \int_if_zero:nT \l_@@_last_col_int
3700   {
3701     \bool_set_true:N \l_@@_last_col_without_value_bool
3702     \int_set:Nn \l_@@_last_col_int { -1 }
3703   }
3704   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3705   \bool_lazy_or:nnT
3706     { \clist_if_empty_p:N \l_@@_vlines_clist }
3707     { \l_@@_except_borders_bool }
3708     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3709   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3710 }
3711 { \endNiceArray }

```

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

```

3712 \cs_new_protected:Npn \@@_NotEmpty:
3713 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

## 15 `{NiceTabular}`, `{NiceTabularX}` and `{NiceTabular*}`

```

3714 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3715 {

```

If the dimension `\l_@@_width_dim` is equal to 0 pt, that means that it has not be set by a previous use of `\NiceMatrixOptions`.

```

3716   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3717     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3718   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3719   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3720   \tl_if_empty:NF \l_@@_short_caption_tl
3721   {
3722     \tl_if_empty:NT \l_@@_caption_tl
3723     {
3724       \@@_error_or_warning:n { short-caption~without~caption }
3725       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3726     }
3727   }
3728   \tl_if_empty:NF \l_@@_label_tl
3729   {
3730     \tl_if_empty:NT \l_@@_caption_tl
3731     { \@@_error_or_warning:n { label~without~caption } }
3732   }
3733   \NewDocumentEnvironment { TabularNote } { b }
3734   {
3735     \bool_if:NTF \l_@@_in_code_after_bool
3736       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3737       {
3738         \tl_if_empty:NF \g_@@_tabularnote_tl
3739         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3740         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3741       }
3742   }
3743   { }
3744   \@@_settings_for_tabular:
3745   \NiceArray { #2 }
3746 }
3747 { \endNiceArray }

```

```

3748 \cs_new_protected:Npn \@@_settings_for_tabular:
3749 {
3750   \bool_set_true:N \l_@@_tabular_bool
3751   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3752   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3753   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3754 }

3755 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3756 {
3757   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3758   \dim_zero_new:N \l_@@_width_dim
3759   \dim_set:Nn \l_@@_width_dim { #1 }
3760   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3761   \@@_settings_for_tabular:
3762   \NiceArray { #3 }
3763 }
3764 {
3765   \endNiceArray
3766   \int_if_zero:nT \g_@@_total_X_weight_int
3767     { \@@_error:n { NiceTabularX~without~X } }
3768 }

3769 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3770 {
3771   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3772   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3773   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3774   \@@_settings_for_tabular:
3775   \NiceArray { #3 }
3776 }
3777 { \endNiceArray }

```

## 16 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`).

```

3778 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3779 {
3780   \bool_lazy_all:nT
3781     {
3782       { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3783       \l_@@_hvlines_bool
3784       { ! \g_@@_delims_bool }
3785       { ! \l_@@_except_borders_bool }
3786     }
3787     {
3788       \bool_set_true:N \l_@@_except_borders_bool
3789       \clist_if_empty:NF \l_@@_corners_clist
3790         { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3791       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3792         {
3793           \@@_stroke_block:nnn
3794             {
3795               rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3796               draw = \l_@@_rules_color_tl
3797             }
3798           { 1-1 }

```

```

3799         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3800     }
3801 }
3802 }

```

```

3803 \cs_new_protected:Npn \@@_after_array:
3804 {
3805     \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.

```

3806     \bool_if:NT \g_@@_last_col_found_bool
3807     { \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`.

```

3808     \bool_if:NT \l_@@_last_col_without_value_bool
3809     { \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.

```

3810     \bool_if:NT \l_@@_last_row_without_value_bool
3811     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3812 \tl_gput_right:Nx \g_@@_aux_tl
3813 {
3814     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3815     {
3816         \int_use:N \l_@@_first_row_int ,
3817         \int_use:N \c@iRow ,
3818         \int_use:N \g_@@_row_total_int ,
3819         \int_use:N \l_@@_first_col_int ,
3820         \int_use:N \c@jCol ,
3821         \int_use:N \g_@@_col_total_int
3822     }
3823 }

```

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`).

```

3824 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3825 {
3826     \tl_gput_right:Nx \g_@@_aux_tl
3827     {
3828         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3829         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3830     }
3831 }
3832 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3833 {
3834     \tl_gput_right:Nx \g_@@_aux_tl
3835     {
3836         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3837         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3838         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3839         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3840     }
3841 }

```

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

```

3842 \@@_create_diag_nodes:

```

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

```

3843 \pgfpicture
3844 \int_step_inline:nn \c@iRow
3845 {
3846   \pgfnodealias
3847   { \@@_env: - ##1 - last }
3848   { \@@_env: - ##1 - \int_use:N \c@jCol }
3849 }
3850 \int_step_inline:nn \c@jCol
3851 {
3852   \pgfnodealias
3853   { \@@_env: - last - ##1 }
3854   { \@@_env: - \int_use:N \c@iRow - ##1 }
3855 }
3856 \str_if_empty:NF \l_@@_name_str
3857 {
3858   \int_step_inline:nn \c@iRow
3859   {
3860     \pgfnodealias
3861     { \l_@@_name_str - ##1 - last }
3862     { \@@_env: - ##1 - \int_use:N \c@jCol }
3863   }
3864   \int_step_inline:nn \c@jCol
3865   {
3866     \pgfnodealias
3867     { \l_@@_name_str - last - ##1 }
3868     { \@@_env: - \int_use:N \c@iRow - ##1 }
3869   }
3870 }
3871 \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.

```

3872 \bool_if:NT \l_@@_parallelize_diags_bool
3873 {
3874   \int_gzero_new:N \g_@@_ddots_int
3875   \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.

```

3876   \dim_gzero_new:N \g_@@_delta_x_one_dim
3877   \dim_gzero_new:N \g_@@_delta_y_one_dim
3878   \dim_gzero_new:N \g_@@_delta_x_two_dim
3879   \dim_gzero_new:N \g_@@_delta_y_two_dim
3880 }
3881 \int_zero_new:N \l_@@_initial_i_int
3882 \int_zero_new:N \l_@@_initial_j_int
3883 \int_zero_new:N \l_@@_final_i_int
3884 \int_zero_new:N \l_@@_final_j_int
3885 \bool_set_false:N \l_@@_initial_open_bool
3886 \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.

```

3887 \bool_if:NT \l_@@_small_bool
3888 {

```

---

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

```

3889 \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3890 \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.

```

3891 \dim_set:Nn \l_@@_xdots_shorten_start_dim
3892 { 0.6 \l_@@_xdots_shorten_start_dim }
3893 \dim_set:Nn \l_@@_xdots_shorten_end_dim
3894 { 0.6 \l_@@_xdots_shorten_end_dim }
3895 }

```

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

```

3896 \@@_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_seq` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3897 \@@_compute_corners:

```

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

```

3898 \@@_adjust_pos_of_blocks_seq:
3899 \@@_deal_with_rounded_corners:
3900 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3901 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3902 \IfPackageLoadedTF { tikz }
3903 {
3904   \tikzset
3905   {
3906     every-picture / .style =
3907     {
3908       overlay ,
3909       remember~picture ,
3910       name~prefix = \@@_env: -
3911     }
3912   }
3913 }
3914 { }
3915 \cs_set_eq:NN \ialign \@@_old_ialign:
3916 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3917 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3918 \cs_set_eq:NN \OverBrace \@@_OverBrace
3919 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3920 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3921 \cs_set_eq:NN \line \@@_line
3922 \g_@@_pre_code_after_tl
3923 \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.

```

3924 \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).

```

3925 \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`).

```

3926 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3927 { \@@_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:`.

```

3928 \bool_set_true:N \l_@@_in_code_after_bool
3929 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3930 \scan_stop:
3931 \tl_gclear:N \g_nicematrix_code_after_tl
3932 \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.

```

3933 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3934 \tl_if_empty:NF \g_@@_pre_code_before_tl
3935 {
3936   \tl_gput_right:Nx \g_@@_aux_tl
3937   {
3938     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3939     { \exp_not:o \g_@@_pre_code_before_tl }
3940   }
3941   \tl_gclear:N \g_@@_pre_code_before_tl
3942 }
3943 \tl_if_empty:NF \g_nicematrix_code_before_tl
3944 {
3945   \tl_gput_right:Nx \g_@@_aux_tl
3946   {
3947     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3948     { \exp_not:o \g_nicematrix_code_before_tl }
3949   }
3950   \tl_gclear:N \g_nicematrix_code_before_tl
3951 }

3952 \str_gclear:N \g_@@_name_env_str
3953 \@@_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`.

```

3954 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3955 }

```

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`.

```

3956 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3957 { \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

---

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

`\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).

```

3958 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3959 {
3960   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3961   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3962 }

```

The following command must *not* be protected.

```

3963 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3964 {
3965   { #1 }
3966   { #2 }
3967   {
3968     \int_compare:nNnTF { #3 } > { 99 }
3969     { \int_use:N \c@iRow }
3970     { #3 }
3971   }
3972   {
3973     \int_compare:nNnTF { #4 } > { 99 }
3974     { \int_use:N \c@jCol }
3975     { #4 }
3976   }
3977   { #5 }
3978 }

```

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).

```

3979 \hook_gput_code:nnn { begindocument } { . }
3980 {
3981   \cs_new_protected:Npx \@@_draw_dotted_lines:
3982   {
3983     \c_@@_pgfortikzpicture_tl
3984     \@@_draw_dotted_lines_i:
3985     \c_@@_endpgfortikzpicture_tl
3986   }
3987 }

```

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

```

3988 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3989 {
3990   \pgfrememberpicturepositiononpagetrue
3991   \pgf@relevantforpicturesizefalse
3992   \g_@@_HVdotsfor_lines_tl
3993   \g_@@_Vdots_lines_tl
3994   \g_@@_Ddots_lines_tl
3995   \g_@@_Iddots_lines_tl
3996   \g_@@_Cdots_lines_tl
3997   \g_@@_Ldots_lines_tl
3998 }

3999 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4000 {
4001   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4002   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4003 }

```

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

```

4004 \pgfdeclareshape { @@_diag_node }
4005 {
4006   \savedanchor { \five }
4007   {
4008     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4009     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4010   }
4011   \anchor { 5 } { \five }
4012   \anchor { center } { \pgfpointorigin }
4013 }

```

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).

```

4014 \cs_new_protected:Npn \@@_create_diag_nodes:
4015 {
4016   \pgfpicture
4017   \pgfrememberpicturepositiononpagetrue
4018   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4019   {
4020     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4021     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4022     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4023     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4024     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4025     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4026     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4027     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4028     \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.

```

4029     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4030     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4031     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4032     \str_if_empty:NF \l_@@_name_str
4033     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4034   }

```

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

```

4035   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4036   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4037   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4038   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4039   \pgfcoordinate
4040   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4041   \pgfnodealias
4042   { \@@_env: - last }
4043   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4044   \str_if_empty:NF \l_@@_name_str
4045   {
4046     \pgfnodealias
4047     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4048     { \@@_env: - \int_use:N \l_tmpa_int }
4049     \pgfnodealias
4050     { \l_@@_name_str - last }
4051     { \@@_env: - last }
4052   }
4053 \endpgfpicture
4054 }

```

## 17 We draw the dotted lines

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

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

The command `\@@_find_extremities_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.

```
4055 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4056 {
```

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

```
4057 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4058 \int_set:Nn \l_@@_initial_i_int { #1 }
4059 \int_set:Nn \l_@@_initial_j_int { #2 }
4060 \int_set:Nn \l_@@_final_i_int { #1 }
4061 \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.

```
4062 \bool_set_false:N \l_@@_stop_loop_bool
4063 \bool_do_until:Nn \l_@@_stop_loop_bool
4064 {
4065   \int_add:Nn \l_@@_final_i_int { #3 }
4066   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4067 \bool_set_false:N \l_@@_final_open_bool
4068 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4069 {
4070   \int_compare:nNnTF { #3 } = \c_one_int
4071   { \bool_set_true:N \l_@@_final_open_bool }
4072   {
4073     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
4074     { \bool_set_true:N \l_@@_final_open_bool }
4075   }
4076 }
4077 {
4078   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
```

```

4079     {
4080         \int_compare:nNnT { #4 } = { -1 }
4081         { \bool_set_true:N \l_@@_final_open_bool }
4082     }
4083     {
4084         \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4085         {
4086             \int_compare:nNnT { #4 } = \c_one_int
4087             { \bool_set_true:N \l_@@_final_open_bool }
4088         }
4089     }
4090 }
4091 \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.

```

4092     {

```

We do a step backwards.

```

4093         \int_sub:Nn \l_@@_final_i_int { #3 }
4094         \int_sub:Nn \l_@@_final_j_int { #4 }
4095         \bool_set_true:N \l_@@_stop_loop_bool
4096     }

```

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`.

```

4097     {
4098         \cs_if_exist:cTF
4099         {
4100             @@ _ dotted _
4101             \int_use:N \l_@@_final_i_int -
4102             \int_use:N \l_@@_final_j_int
4103         }
4104         {
4105             \int_sub:Nn \l_@@_final_i_int { #3 }
4106             \int_sub:Nn \l_@@_final_j_int { #4 }
4107             \bool_set_true:N \l_@@_final_open_bool
4108             \bool_set_true:N \l_@@_stop_loop_bool
4109         }
4110         {
4111             \cs_if_exist:cTF
4112             {
4113                 pgf @ sh @ ns @ \@@_env:
4114                 - \int_use:N \l_@@_final_i_int
4115                 - \int_use:N \l_@@_final_j_int
4116             }
4117             { \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.

```

4118         {
4119             \cs_set:cpn
4120             {
4121                 @@ _ dotted _
4122                 \int_use:N \l_@@_final_i_int -
4123                 \int_use:N \l_@@_final_j_int
4124             }
4125             { }
4126         }
4127     }
4128 }
4129 }

```

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

```

4130 \bool_set_false:N \l_@@_stop_loop_bool
4131 \bool_do_until:Nn \l_@@_stop_loop_bool
4132 {
4133   \int_sub:Nn \l_@@_initial_i_int { #3 }
4134   \int_sub:Nn \l_@@_initial_j_int { #4 }
4135   \bool_set_false:N \l_@@_initial_open_bool
4136   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4137   {
4138     \int_compare:nNnTF { #3 } = \c_one_int
4139     { \bool_set_true:N \l_@@_initial_open_bool }
4140     {
4141       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4142       { \bool_set_true:N \l_@@_initial_open_bool }
4143     }
4144   }
4145   {
4146     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4147     {
4148       \int_compare:nNnT { #4 } = \c_one_int
4149       { \bool_set_true:N \l_@@_initial_open_bool }
4150     }
4151     {
4152       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4153       {
4154         \int_compare:nNnT { #4 } = { -1 }
4155         { \bool_set_true:N \l_@@_initial_open_bool }
4156       }
4157     }
4158   }
4159   \bool_if:NTF \l_@@_initial_open_bool
4160   {
4161     \int_add:Nn \l_@@_initial_i_int { #3 }
4162     \int_add:Nn \l_@@_initial_j_int { #4 }
4163     \bool_set_true:N \l_@@_stop_loop_bool
4164   }
4165   {
4166     \cs_if_exist:cTF
4167     {
4168       @@ _ dotted _
4169       \int_use:N \l_@@_initial_i_int -
4170       \int_use:N \l_@@_initial_j_int
4171     }
4172     {
4173       \int_add:Nn \l_@@_initial_i_int { #3 }
4174       \int_add:Nn \l_@@_initial_j_int { #4 }
4175       \bool_set_true:N \l_@@_initial_open_bool
4176       \bool_set_true:N \l_@@_stop_loop_bool
4177     }
4178     {
4179       \cs_if_exist:cTF
4180       {
4181         pgf @ sh @ ns @ \@@_env:
4182         - \int_use:N \l_@@_initial_i_int
4183         - \int_use:N \l_@@_initial_j_int
4184       }
4185       { \bool_set_true:N \l_@@_stop_loop_bool }
4186       {
4187         \cs_set:cpn
4188         {
4189           @@ _ dotted _
4190           \int_use:N \l_@@_initial_i_int -

```

```

4191             \int_use:N \l_@@_initial_j_int
4192         }
4193     { }
4194 }
4195 }
4196 }
4197 }

```

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.

```

4198 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4199 {
4200     { \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`.

```

4201     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4202     { \int_use:N \l_@@_final_i_int }
4203     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4204     { } % for the name of the block
4205 }
4206 }

```

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.

```

4207 \cs_new_protected:Npn \@@_open_shorten:
4208 {
4209     \bool_if:NT \l_@@_initial_open_bool
4210     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4211     \bool_if:NT \l_@@_final_open_bool
4212     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4213 }

```

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).

```

4214 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4215 {
4216     \int_set:Nn \l_@@_row_min_int 1
4217     \int_set:Nn \l_@@_col_min_int 1
4218     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4219     \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`.

```

4220 \seq_map_inline:Nn \g_@@_submatrix_seq
4221 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4222 }

```

#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.

```

4223 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4224 {
4225     \int_compare:nNnF { #3 } > { #1 }
4226     {
4227         \int_compare:nNnF { #1 } > { #5 }
4228         {

```

```

4229         \int_compare:nNnF { #4 } > { #2 }
4230         {
4231             \int_compare:nNnF { #2 } > { #6 }
4232             {
4233                 \int_set:Nn \l_@@_row_min_int
4234                 { \int_max:nn \l_@@_row_min_int { #3 } }
4235                 \int_set:Nn \l_@@_col_min_int
4236                 { \int_max:nn \l_@@_col_min_int { #4 } }
4237                 \int_set:Nn \l_@@_row_max_int
4238                 { \int_min:nn \l_@@_row_max_int { #5 } }
4239                 \int_set:Nn \l_@@_col_max_int
4240                 { \int_min:nn \l_@@_col_max_int { #6 } }
4241             }
4242         }
4243     }
4244 }
4245 }

4246 \cs_new_protected:Npn \@@_set_initial_coords:
4247 {
4248     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4249     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4250 }
4251 \cs_new_protected:Npn \@@_set_final_coords:
4252 {
4253     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4254     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4255 }
4256 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4257 {
4258     \pgfpointanchor
4259     {
4260         \@@_env:
4261         - \int_use:N \l_@@_initial_i_int
4262         - \int_use:N \l_@@_initial_j_int
4263     }
4264     { #1 }
4265     \@@_set_initial_coords:
4266 }
4267 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4268 {
4269     \pgfpointanchor
4270     {
4271         \@@_env:
4272         - \int_use:N \l_@@_final_i_int
4273         - \int_use:N \l_@@_final_j_int
4274     }
4275     { #1 }
4276     \@@_set_final_coords:
4277 }
4278 \cs_new_protected:Npn \@@_open_x_initial_dim:
4279 {
4280     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4281     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4282     {
4283         \cs_if_exist:cT
4284         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4285         {
4286             \pgfpointanchor
4287             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4288             { west }
4289             \dim_set:Nn \l_@@_x_initial_dim
4290             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }

```

```

4291     }
4292 }

```

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

```

4293 \dim_compare:nNt \l_@@_x_initial_dim = \c_max_dim
4294 {
4295   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4296   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4297   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4298 }
4299 }

4300 \cs_new_protected:Npn \@@_open_x_final_dim:
4301 {
4302   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4303   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4304   {
4305     \cs_if_exist:cT
4306     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4307     {
4308       \pgfpoinanchor
4309       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4310       { east }
4311       \dim_set:Nn \l_@@_x_final_dim
4312       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4313     }
4314   }

```

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

```

4315 \dim_compare:nNt \l_@@_x_final_dim = { - \c_max_dim }
4316 {
4317   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4318   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4319   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4320 }
4321 }

```

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.

```

4322 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4323 {
4324   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4325   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4326   {
4327     \@@_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.

```

4328   \group_begin:
4329   \@@_open_shorten:
4330   \int_if_zero:nTF { #1 }
4331   { \color { nicematrix-first-row } }
4332   {

```

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.

```

4333     \int_compare:nNt { #1 } = \l_@@_last_row_int
4334     { \color { nicematrix-last-row } }
4335   }
4336   \keys_set:nn { NiceMatrix / xdots } { #3 }
4337   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4338   \@@_actually_draw_Ldots:
4339   \group_end:
4340 }
4341 }

```

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`.

```

4342 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4343 {
4344   \bool_if:NTF \l_@@_initial_open_bool
4345   {
4346     \@@_open_x_initial_dim:
4347     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4348     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4349   }
4350   { \@@_set_initial_coords_from_anchor:n { base~east } }
4351   \bool_if:NTF \l_@@_final_open_bool
4352   {
4353     \@@_open_x_final_dim:
4354     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4355     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4356   }
4357   { \@@_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.

```

4358   \bool_lazy_all:nTF
4359   {
4360     \l_@@_initial_open_bool
4361     \l_@@_final_open_bool
4362     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4363   }
4364   {
4365     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4366     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4367   }

```

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 (?).

```

4368   {
4369     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4370     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4371   }
4372   \@@_draw_line:
4373 }

```

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.

```

4374 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4375 {
4376   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4377   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4378   {
4379     \@@_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.

```

4380     \group_begin:
4381     \@@_open_shorten:
4382     \int_if_zero:nTF { #1 }
4383     { \color { nicematrix-first-row } }
4384     {

```

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.

```

4385         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4386         { \color { nicematrix-last-row } }
4387     }
4388     \keys_set:nn { NiceMatrix / xdots } { #3 }
4389     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4390     \@@_actually_draw_Cdots:
4391 \group_end:
4392 }
4393 }

```

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`.

```

4394 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4395 {
4396     \bool_if:NTF \l_@@_initial_open_bool
4397     { \@@_open_x_initial_dim: }
4398     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4399     \bool_if:NTF \l_@@_final_open_bool
4400     { \@@_open_x_final_dim: }
4401     { \@@_set_final_coords_from_anchor:n { mid-west } }
4402     \bool_lazy_and:nnTF
4403     \l_@@_initial_open_bool
4404     \l_@@_final_open_bool
4405     {
4406         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4407         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4408         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4409         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4410         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4411     }
4412     {
4413         \bool_if:NT \l_@@_initial_open_bool
4414         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4415         \bool_if:NT \l_@@_final_open_bool
4416         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4417     }
4418     \@@_draw_line:
4419 }
4420 \cs_new_protected:Npn \@@_open_y_initial_dim:
4421 {
4422     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4423     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4424     {

```

```

4425 \cs_if_exist:cT
4426 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4427 {
4428     \pgfpointanchor
4429     { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4430     { north }
4431     \dim_set:Nn \l_@@_y_initial_dim
4432     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4433 }
4434 }
4435 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4436 {
4437     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4438     \dim_set:Nn \l_@@_y_initial_dim
4439     {
4440         \fp_to_dim:n
4441         {
4442             \pgf@y
4443             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4444         }
4445     }
4446 }
4447 }
4448 \cs_new_protected:Npn \@@_open_y_final_dim:
4449 {
4450     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4451     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4452     {
4453         \cs_if_exist:cT
4454         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4455         {
4456             \pgfpointanchor
4457             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4458             { south }
4459             \dim_set:Nn \l_@@_y_final_dim
4460             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4461         }
4462     }
4463     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4464     {
4465         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4466         \dim_set:Nn \l_@@_y_final_dim
4467         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4468     }
4469 }

```

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.

```

4470 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4471 {
4472     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4473     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4474     {
4475         \@@_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.

```

4476 \group_begin:
4477     \@@_open_shorten:
4478     \int_if_zero:nTF { #2 }
4479     { \color { nicematrix-first-col } }
4480     {
4481         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4482         { \color { nicematrix-last-col } }

```

```

4483     }
4484     \keys_set:nn { NiceMatrix / xdots } { #3 }
4485     \tl_if_empty:oF \l_@@_xdots_color_tl
4486     { \color { \l_@@_xdots_color_tl } }
4487     \@@_actually_draw_Vdots:
4488     \group_end:
4489   }
4490 }

```

The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool.`

The following function is also used by `\Vdotsfor`.

```

4491 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4492 {

```

First, the case of a dotted line open on both sides.

```

4493   \bool_lazy_and:nnTF \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.

```

4494   {
4495     \@@_open_y_initial_dim:
4496     \@@_open_y_final_dim:
4497     \int_if_zero:nTF \l_@@_initial_j_int

```

We have a dotted line open on both sides in the “first column”.

```

4498     {
4499       \@@_qpoint:n { col - 1 }
4500       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4501       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4502       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4503       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4504     }
4505     {
4506       \bool_lazy_and:nnTF
4507       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4508       { \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”.

```

4509     {
4510       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4511       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4512       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4513       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4514       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4515     }

```

We have a dotted line open on both sides which is *not* in an exterior column.

```

4516     {
4517       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4518       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4519       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4520       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4521     }
4522   }
4523 }

```

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 `l` or may be considered as if.

```

4524 {
4525     \bool_set_false:N \l_tmpa_bool
4526     \bool_if:NF \l_@@_initial_open_bool
4527     {
4528         \bool_if:NF \l_@@_final_open_bool
4529         {
4530             \@@_set_initial_coords_from_anchor:n { south-west }
4531             \@@_set_final_coords_from_anchor:n { north-west }
4532             \bool_set:Nn \l_tmpa_bool
4533             { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4534         }
4535     }

```

Now, we try to determine whether the column is of type `c` or may be considered as if.

```

4536     \bool_if:NTF \l_@@_initial_open_bool
4537     {
4538         \@@_open_y_initial_dim:
4539         \@@_set_final_coords_from_anchor:n { north }
4540         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4541     }
4542     {
4543         \@@_set_initial_coords_from_anchor:n { south }
4544         \bool_if:NTF \l_@@_final_open_bool
4545         \@@_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.

```

4546     {
4547         \@@_set_final_coords_from_anchor:n { north }
4548         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4549         {
4550             \dim_set:Nn \l_@@_x_initial_dim
4551             {
4552                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4553                 \l_@@_x_initial_dim \l_@@_x_final_dim
4554             }
4555         }
4556     }
4557 }
4558 }
4559 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4560 \@@_draw_line:
4561 }

```

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.

```

4562 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4563 {
4564     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4565     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4566     {
4567         \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4568     \group_begin:
4569     \@@_open_shorten:

```

```

4570         \keys_set:nn { NiceMatrix / xdots } { #3 }
4571         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4572         \@@_actually_draw_Ddots:
4573     \group_end:
4574 }
4575 }

```

The command `\@@_actually_draw_Ddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4576 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4577 {
4578     \bool_if:NTF \l_@@_initial_open_bool
4579     {
4580         \@@_open_y_initial_dim:
4581         \@@_open_x_initial_dim:
4582     }
4583     { \@@_set_initial_coords_from_anchor:n { south~east } }
4584     \bool_if:NTF \l_@@_final_open_bool
4585     {
4586         \@@_open_x_final_dim:
4587         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4588     }
4589     { \@@_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.

```

4590     \bool_if:NT \l_@@_parallelize_diags_bool
4591     {
4592         \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).

```

4593     \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.

```

4594     {
4595         \dim_gset:Nn \g_@@_delta_x_one_dim
4596         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4597         \dim_gset:Nn \g_@@_delta_y_one_dim
4598         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4599     }

```

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`.

```

4600     {
4601         \dim_set:Nn \l_@@_y_final_dim
4602         {
4603             \l_@@_y_initial_dim +
4604             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4605             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4606         }
4607     }
4608 }
4609 \@@_draw_line:
4610 }

```

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.

```

4611 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4612 {
4613   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4614   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4615   {
4616     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4617     \group_begin:
4618     \@@_open_shorten:
4619     \keys_set:nn { NiceMatrix / xdots } { #3 }
4620     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4621     \@@_actually_draw_Iddots:
4622   \group_end:
4623 }
4624 }

```

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`.

```

4625 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4626 {
4627   \bool_if:NTF \l_@@_initial_open_bool
4628   {
4629     \@@_open_y_initial_dim:
4630     \@@_open_x_initial_dim:
4631   }
4632   { \@@_set_initial_coords_from_anchor:n { south-west } }
4633   \bool_if:NTF \l_@@_final_open_bool
4634   {
4635     \@@_open_y_final_dim:
4636     \@@_open_x_final_dim:
4637   }
4638   { \@@_set_final_coords_from_anchor:n { north-east } }
4639   \bool_if:NT \l_@@_parallelize_diags_bool
4640   {
4641     \int_gincr:N \g_@@_iddots_int
4642     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4643     {
4644       \dim_gset:Nn \g_@@_delta_x_two_dim
4645       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4646       \dim_gset:Nn \g_@@_delta_y_two_dim
4647       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4648     }
4649     {
4650       \dim_set:Nn \l_@@_y_final_dim
4651       {
4652         \l_@@_y_initial_dim +
4653         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4654         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim

```

```

4655         }
4656     }
4657 }
4658 \@@_draw_line:
4659 }

```

## 18 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`

```

4660 \cs_new_protected:Npn \@@_draw_line:
4661 {
4662     \pgfrememberpicturepositiononpagetrue
4663     \pgf@relevantforpicturesizefalse
4664     \bool_lazy_or:nnTF
4665     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4666     \l_@@_dotted_bool
4667     \@@_draw_standard_dotted_line:
4668     \@@_draw_unstandard_dotted_line:
4669 }

```

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.

```

4670 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4671 {
4672     \begin { scope }
4673     \@@_draw_unstandard_dotted_line:o
4674     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4675 }

```

We have used the fact that, in PGF, an 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.

```

4676 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4677 {
4678     \@@_draw_unstandard_dotted_line:nooo
4679     { #1 }
4680     \l_@@_xdots_up_tl
4681     \l_@@_xdots_down_tl
4682     \l_@@_xdots_middle_tl
4683 }
4684 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4685 \hook_gput_code:nnn { begindocument } { . }
4686 {
4687   \IfPackageLoadedTF { tikz }
4688   {
4689     \tikzset
4690     {
4691       @@_node_above / .style = { sloped , above } ,
4692       @@_node_below / .style = { sloped , below } ,
4693       @@_node_middle / .style =
4694       {
4695         sloped ,
4696         inner-sep = \c_@@_innersep_middle_dim
4697       }
4698     }
4699   }
4700   { }
4701 }

4702 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4703 {

```

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.

```

4704 \dim_zero_new:N \l_@@_l_dim
4705 \dim_set:Nn \l_@@_l_dim
4706 {
4707   \fp_to_dim:n
4708   {
4709     sqrt
4710     (
4711       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4712       +
4713       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4714     )
4715   }
4716 }

```

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.

```

4717 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4718 {
4719   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4720   \@@_draw_unstandard_dotted_line_i:
4721 }

```

If the key `xdots/horizontal-labels` has been used.

```

4722 \bool_if:NT \l_@@_xdots_h_labels_bool
4723 {
4724   \tikzset
4725   {
4726     @@_node_above / .style = { auto = left } ,
4727     @@_node_below / .style = { auto = right } ,
4728     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4729   }
4730 }

```

```

4731 \tl_if_empty:nF { #4 }
4732 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4733 \draw
4734 [ #1 ]
4735 ( \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).

```

4736 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4737 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4738 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4739 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4740 \end { scope }
4741 }
4742 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4743 {
4744 \dim_set:Nn \l_tmpa_dim
4745 {
4746 \l_@@_x_initial_dim
4747 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4748 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4749 }
4750 \dim_set:Nn \l_tmpb_dim
4751 {
4752 \l_@@_y_initial_dim
4753 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4754 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4755 }
4756 \dim_set:Nn \l_@@_tmpc_dim
4757 {
4758 \l_@@_x_final_dim
4759 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4760 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4761 }
4762 \dim_set:Nn \l_@@_tmpd_dim
4763 {
4764 \l_@@_y_final_dim
4765 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4766 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4767 }
4768 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4769 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4770 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4771 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4772 }
4773 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4774 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4775 {
4776 \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.

```

4777 \dim_zero_new:N \l_@@_l_dim
4778 \dim_set:Nn \l_@@_l_dim
4779 {
4780 \fp_to_dim:n
4781 {
4782 sqrt
4783 (

```

```

4784      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4785      +
4786      ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4787    )
4788  }
4789 }

```

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.

```

4790 \dim_compare:nNt \l_@@_l_dim < \c_@@_max_l_dim
4791 {
4792   \dim_compare:nNt \l_@@_l_dim > { 1 pt }
4793   \@@_draw_standard_dotted_line_i:
4794 }
4795 \group_end:
4796 \bool_lazy_all:nF
4797 {
4798   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4799   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4800   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4801 }
4802 \l_@@_labels_standard_dotted_line:
4803 }
4804 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4805 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4806 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4807 \int_set:Nn \l_tmpa_int
4808 {
4809   \dim_ratio:nn
4810   {
4811     \l_@@_l_dim
4812     - \l_@@_xdots_shorten_start_dim
4813     - \l_@@_xdots_shorten_end_dim
4814   }
4815   \l_@@_xdots_inter_dim
4816 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4817 \dim_set:Nn \l_tmpa_dim
4818 {
4819   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4820   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4821 }
4822 \dim_set:Nn \l_tmpb_dim
4823 {
4824   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4825   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4826 }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4827 \dim_gadd:Nn \l_@@_x_initial_dim
4828 {
4829   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4830   \dim_ratio:nn
4831   {
4832     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int

```

```

4833         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4834     }
4835     { 2 \l_@@_l_dim }
4836 }
4837 \dim_gadd:Nn \l_@@_y_initial_dim
4838 {
4839     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4840     \dim_ratio:nn
4841     {
4842         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4843         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4844     }
4845     { 2 \l_@@_l_dim }
4846 }
4847 \pgf@relevantforpicturesizefalse
4848 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4849 {
4850     \pgfpathcircle
4851     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4852     { \l_@@_xdots_radius_dim }
4853     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4854     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4855 }
4856 \pgfusepathqfill
4857 }

4858 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4859 {
4860     \pgfscope
4861     \pgftransformshift
4862     {
4863         \pgfpointlineattime { 0.5 }
4864         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4865         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4866     }
4867     \fp_set:Nn \l_tmpa_fp
4868     {
4869         atand
4870         (
4871             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4872             \l_@@_x_final_dim - \l_@@_x_initial_dim
4873         )
4874     }
4875     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4876     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4877     \tl_if_empty:NF \l_@@_xdots_middle_tl
4878     {
4879         \begin { pgfscope }
4880         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4881         \pgfnode
4882         { rectangle }
4883         { center }
4884         {
4885             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4886             {
4887                 \c_math_toggle_token
4888                 \scriptstyle \l_@@_xdots_middle_tl
4889                 \c_math_toggle_token
4890             }
4891         }
4892         { }
4893         {
4894             \pgfsetfillcolor { white }

```

```

4895         \pgfusepath { fill }
4896     }
4897     \end { pgfscope }
4898 }
4899 \tl_if_empty:NF \l_@@_xdots_up_tl
4900 {
4901     \pgfnode
4902     { rectangle }
4903     { south }
4904     {
4905         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4906         {
4907             \c_math_toggle_token
4908             \scriptstyle \l_@@_xdots_up_tl
4909             \c_math_toggle_token
4910         }
4911     }
4912     { }
4913     { \pgfusepath { } }
4914 }
4915 \tl_if_empty:NF \l_@@_xdots_down_tl
4916 {
4917     \pgfnode
4918     { rectangle }
4919     { north }
4920     {
4921         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4922         {
4923             \c_math_toggle_token
4924             \scriptstyle \l_@@_xdots_down_tl
4925             \c_math_toggle_token
4926         }
4927     }
4928     { }
4929     { \pgfusepath { } }
4930 }
4931 \endpgfscope
4932 }

```

## 19 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.

```

4933 \hook_gput_code:nnn { begindocument } { . }
4934 {
4935     \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4936     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4937     \cs_new_protected:Npn \@@_Ldots
4938     { \@@_collect_options:n { \@@_Ldots_i } }
4939     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4940     {
4941         \int_if_zero:nTF \c@jCol

```

```

4942     { \@@_error:nn { in~first~col } \Ldots }
4943     {
4944       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4945       { \@@_error:nn { in~last~col } \Ldots }
4946       {
4947         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4948         { #1 , down = #2 , up = #3 , middle = #4 }
4949       }
4950     }
4951     \bool_if:NF \l_@@_nullify_dots_bool
4952     { \phantom { \ensuremath { \@@_old_ldots } } }
4953     \bool_gset_true:N \g_@@_empty_cell_bool
4954   }

4955 \cs_new_protected:Npn \@@_Cdots
4956 { \@@_collect_options:n { \@@_Cdots_i } }
4957 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4958 {
4959   \int_if_zero:nTF \c@jCol
4960   { \@@_error:nn { in~first~col } \Cdots }
4961   {
4962     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4963     { \@@_error:nn { in~last~col } \Cdots }
4964     {
4965       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4966       { #1 , down = #2 , up = #3 , middle = #4 }
4967     }
4968   }
4969   \bool_if:NF \l_@@_nullify_dots_bool
4970   { \phantom { \ensuremath { \@@_old_cdots } } }
4971   \bool_gset_true:N \g_@@_empty_cell_bool
4972 }

4973 \cs_new_protected:Npn \@@_Vdots
4974 { \@@_collect_options:n { \@@_Vdots_i } }
4975 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4976 {
4977   \int_if_zero:nTF \c@iRow
4978   { \@@_error:nn { in~first~row } \Vdots }
4979   {
4980     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4981     { \@@_error:nn { in~last~row } \Vdots }
4982     {
4983       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4984       { #1 , down = #2 , up = #3 , middle = #4 }
4985     }
4986   }
4987   \bool_if:NF \l_@@_nullify_dots_bool
4988   { \phantom { \ensuremath { \@@_old_vdots } } }
4989   \bool_gset_true:N \g_@@_empty_cell_bool
4990 }

4991 \cs_new_protected:Npn \@@_Ddots
4992 { \@@_collect_options:n { \@@_Ddots_i } }
4993 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4994 {
4995   \int_case:nnF \c@iRow
4996   {
4997     0 { \@@_error:nn { in~first~row } \Ddots }
4998     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4999   }

```

```

5000 {
5001   \int_case:nnF \c@jCol
5002   {
5003     0 { \@@_error:nn { in~first~col } \Ddots }
5004     \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5005   }
5006   {
5007     \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5008     \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5009     { #1 , down = #2 , up = #3 , middle = #4 }
5010   }
5011 }
5012 }
5013 \bool_if:NF \l_@@_nullify_dots_bool
5014 { \phantom { \ensuremath { \@@_old_ddots } } }
5015 \bool_gset_true:N \g_@@_empty_cell_bool
5016 }

5017 \cs_new_protected:Npn \@@_Iddots
5018 { \@@_collect_options:n { \@@_Iddots_i } }
5019 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5020 {
5021   \int_case:nnF \c@iRow
5022   {
5023     0 { \@@_error:nn { in~first~row } \Iddots }
5024     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5025   }
5026   {
5027     \int_case:nnF \c@jCol
5028     {
5029       0 { \@@_error:nn { in~first~col } \Iddots }
5030       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5031     }
5032     {
5033       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5034       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5035       { #1 , down = #2 , up = #3 , middle = #4 }
5036     }
5037   }
5038   \bool_if:NF \l_@@_nullify_dots_bool
5039   { \phantom { \ensuremath { \@@_old_iddots } } }
5040   \bool_gset_true:N \g_@@_empty_cell_bool
5041 }
5042 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5043 \keys_define:nn { NiceMatrix / Ddots }
5044 {
5045   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5046   draw-first .default:n = true ,
5047   draw-first .value_forbidden:n = true
5048 }

```

The command \@@\_Hspace: will be linked to \hspace in {NiceArray}.

```

5049 \cs_new_protected:Npn \@@_Hspace:
5050 {
5051   \bool_gset_true:N \g_@@_empty_cell_bool
5052   \hspace
5053 }

```

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`.

```
5054 \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`.

```
5055 \cs_new:Npn \@@_Hdotsfor:
5056 {
5057   \bool_lazy_and:nnTF
5058     { \int_if_zero_p:n \c@jCol }
5059     { \int_if_zero_p:n \l_@@_first_col_int }
5060     {
5061       \bool_if:NTF \g_@@_after_col_zero_bool
5062       {
5063         \multicolumn { 1 } { c } { }
5064         \@@_Hdotsfor_i
5065       }
5066       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5067     }
5068     {
5069       \multicolumn { 1 } { c } { }
5070       \@@_Hdotsfor_i
5071     }
5072 }
```

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:`).

```
5073 \hook_gput_code:nnn { begindocument } { . }
5074 {
5075   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5076   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```
5077   \cs_new_protected:Npn \@@_Hdotsfor_i
5078     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5079   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5080     {
5081       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5082       {
5083         \@@_Hdotsfor:nnnn
5084         { \int_use:N \c@iRow }
5085         { \int_use:N \c@jCol }
5086         { #2 }
5087         {
5088           #1 , #3 ,
5089           down = \exp_not:n { #4 } ,
5090           up = \exp_not:n { #5 } ,
5091           middle = \exp_not:n { #6 }
5092         }
5093       }
5094       \prg_replicate:nn { #2 - 1 }
5095       {
5096         &
5097         \multicolumn { 1 } { c } { }
5098         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5099       }
5100     }
5101 }
```

```

5102 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5103 {
5104   \bool_set_false:N \l_@@_initial_open_bool
5105   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5106   \int_set:Nn \l_@@_initial_i_int { #1 }
5107   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5108   \int_compare:nNnTF { #2 } = \c_one_int
5109   {
5110     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5111     \bool_set_true:N \l_@@_initial_open_bool
5112   }
5113   {
5114     \cs_if_exist:cTF
5115     {
5116       pgf @ sh @ ns @ \@@_env:
5117       - \int_use:N \l_@@_initial_i_int
5118       - \int_eval:n { #2 - 1 }
5119     }
5120     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5121     {
5122       \int_set:Nn \l_@@_initial_j_int { #2 }
5123       \bool_set_true:N \l_@@_initial_open_bool
5124     }
5125   }
5126   \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5127   {
5128     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5129     \bool_set_true:N \l_@@_final_open_bool
5130   }
5131   {
5132     \cs_if_exist:cTF
5133     {
5134       pgf @ sh @ ns @ \@@_env:
5135       - \int_use:N \l_@@_final_i_int
5136       - \int_eval:n { #2 + #3 }
5137     }
5138     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5139     {
5140       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5141       \bool_set_true:N \l_@@_final_open_bool
5142     }
5143   }
5144   \group_begin:
5145   \@@_open_shorten:
5146   \int_if_zero:nTF { #1 }
5147   { \color { nicematrix-first-row } }
5148   {
5149     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5150     { \color { nicematrix-last-row } }
5151   }
5152
5153   \keys_set:nn { NiceMatrix / xdots } { #4 }
5154   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5155   \@@_actually_draw_Ldots:
5156   \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`).

```

5157 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5158 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5159 }

5160 \hook_gput_code:nnn { begindocument } { . }
5161 {
5162   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5163   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5164   \cs_new_protected:Npn \@@_Vdotsfor:
5165     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5166   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5167     {
5168       \bool_gset_true:N \g_@@_empty_cell_bool
5169       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5170         {
5171           \@@_Vdotsfor:nnnn
5172           { \int_use:N \c@iRow }
5173           { \int_use:N \c@jCol }
5174           { #2 }
5175           {
5176             #1 , #3 ,
5177             down = \exp_not:n { #4 } ,
5178             up = \exp_not:n { #5 } ,
5179             middle = \exp_not:n { #6 }
5180           }
5181         }
5182     }
5183 }

```

```

5184 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5185 {
5186   \bool_set_false:N \l_@@_initial_open_bool
5187   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5188 \int_set:Nn \l_@@_initial_j_int { #2 }
5189 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5190 \int_compare:nNnTF { #1 } = \c_one_int
5191 {
5192   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5193   \bool_set_true:N \l_@@_initial_open_bool
5194 }
5195 {
5196   \cs_if_exist:cTF
5197   {
5198     pgf @ sh @ ns @ \@@_env:
5199     - \int_eval:n { #1 - 1 }
5200     - \int_use:N \l_@@_initial_j_int
5201   }
5202   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5203   {
5204     \int_set:Nn \l_@@_initial_i_int { #1 }
5205     \bool_set_true:N \l_@@_initial_open_bool
5206   }
5207 }
5208 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5209 {
5210   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5211   \bool_set_true:N \l_@@_final_open_bool
5212 }
5213 {

```

```

5214 \cs_if_exist:cTF
5215 {
5216   pgf @ sh @ ns @ \@@_env:
5217   - \int_eval:n { #1 + #3 }
5218   - \int_use:N \l_@@_final_j_int
5219 }
5220 { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5221 {
5222   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5223   \bool_set_true:N \l_@@_final_open_bool
5224 }
5225 }
5226 \group_begin:
5227 \@@_open_shorten:
5228 \int_if_zero:nTF { #2 }
5229 { \color { nicematrix-first-col } }
5230 {
5231   \int_compare:nNt { #2 } = \g_@@_col_total_int
5232   { \color { nicematrix-last-col } }
5233 }
5234 \keys_set:nn { NiceMatrix / xdots } { #4 }
5235 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5236 \@@_actually_draw_Vdots:
5237 \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5238 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5239 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5240 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5241 \NewDocumentCommand \@@_rotate: { 0 { } }
5242 {
5243   \peek_remove_spaces:n
5244   {
5245     \bool_gset_true:N \g_@@_rotate_bool
5246     \keys_set:nn { NiceMatrix / rotate } { #1 }
5247   }
5248 }
5249 \keys_define:nn { NiceMatrix / rotate }
5250 {
5251   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5252   c .value_forbidden:n = true ,
5253   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5254 }

```

## 20 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>

```

5255 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5256 {
5257   \tl_if_empty:nTF { #2 }
5258   { #1 }
5259   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5260 }
5261 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5262 { \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).

```

5263 \hook_gput_code:nnn { begindocument } { . }
5264 {
5265   \cs_set_nopar:Npn \l_@@_argspec_tl
5266   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5267   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5268   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5269   {
5270     \group_begin:
5271     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5272     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5273     \use:e
5274     {
5275       \@@_line_i:nn
5276       { \@@_double_int_eval:n #2 - \q_stop }
5277       { \@@_double_int_eval:n #3 - \q_stop }
5278     }
5279     \group_end:
5280   }
5281 }
5282 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5283 {
5284   \bool_set_false:N \l_@@_initial_open_bool
5285   \bool_set_false:N \l_@@_final_open_bool
5286   \bool_lazy_or:nnTF
5287   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5288   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5289   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5290   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5291 }
5292 \hook_gput_code:nnn { begindocument } { . }
5293 {
5294   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5295   {
```

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:`.

```

5296   \c_@@_pgfortikzpicture_tl
5297   \@@_draw_line_iii:nn { #1 } { #2 }
```

---

<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`.

```

5298         \c_@@_endpgfortikzpicture_tl
5299     }
5300 }

```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```

5301 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5302 {
5303     \pgfrememberpicturepositiononpagetrue
5304     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5305     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5306     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5307     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5308     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5309     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5310     \@@_draw_line:
5311 }

```

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).

## 21 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`

```

5312 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5313 { \int_compare:nNnT \c@iRow < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```

5314 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5315 {
5316     \tl_gput_right:Nx \g_@@_row_style_tl
5317     {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5318         \exp_not:N
5319         \@@_if_row_less_than:nn
5320         { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5321         { \exp_not:n { #1 } }
5322     }
5323 }
5324 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5325 \keys_define:nn { NiceMatrix / RowStyle }
5326 {
5327     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5328     cell-space-top-limit .value_required:n = true ,
5329     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5330     cell-space-bottom-limit .value_required:n = true ,

```

```

5331 cell-space-limits .meta:n =
5332 {
5333     cell-space-top-limit = #1 ,
5334     cell-space-bottom-limit = #1 ,
5335 } ,
5336 color .tl_set:N = \l_@@_color_tl ,
5337 color .value_required:n = true ,
5338 bold .bool_set:N = \l_@@_bold_row_style_bool ,
5339 bold .default:n = true ,
5340 nb-rows .code:n =
5341     \str_if_eq:nnTF { #1 } { * }
5342     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5343     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5344 nb-rows .value_required:n = true ,
5345 rowcolor .tl_set:N = \l_tmpa_tl ,
5346 rowcolor .value_required:n = true ,
5347 rowcolor .initial:n = ,
5348 unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
5349 }

```

```

5350 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5351 {
5352     \group_begin:
5353     \tl_clear:N \l_tmpa_tl % value of \rowcolor
5354     \tl_clear:N \l_@@_color_tl
5355     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5356     \dim_zero:N \l_tmpa_dim
5357     \dim_zero:N \l_tmpb_dim
5358     \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5359     \tl_if_empty:NF \l_tmpa_tl
5360     {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5361         \tl_gput_right:Nx \g_@@_pre_code_before_tl
5362         {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5363             \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5364             { \int_use:N \c@iRow - \int_use:N \c@jCol }
5365             { \int_use:N \c@iRow - * }
5366         }

```

Then, the other rows (if there is several rows).

```

5367         \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5368         {
5369             \tl_gput_right:Nx \g_@@_pre_code_before_tl
5370             {
5371                 \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5372                 {
5373                     \int_eval:n { \c@iRow + 1 }
5374                     - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5375                 }
5376             }
5377         }
5378     }
5379     \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5380     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5381     {
5382         \exp_args:Nx \@@_put_in_row_style:n
5383         {
5384             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5385             {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5386         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5387         { \dim_use:N \l_tmpa_dim }
5388     }
5389 }
5390 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5391     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5392     {
5393         \exp_args:Nx \@@_put_in_row_style:n
5394         {
5395             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5396             {
5397                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5398                 { \dim_use:N \l_tmpb_dim }
5399             }
5400         }
5401     }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5402     \tl_if_empty:NF \l_@@_color_tl
5403     {
5404         \@@_put_in_row_style:e
5405         {
5406             \mode_leave_vertical:
5407             \@@_color:n { \l_@@_color_tl }
5408         }
5409     }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5410     \bool_if:NT \l_@@_bold_row_style_bool
5411     {
5412         \@@_put_in_row_style:n
5413         {
5414             \exp_not:n
5415             {
5416                 \if_mode_math:
5417                 \c_math_toggle_token
5418                 \bfseries \boldmath
5419                 \c_math_toggle_token
5420             }
5421             \else:
5422             \bfseries \boldmath
5423             \fi:
5424         }
5425     }
5426     \group_end:
5427     \g_@@_row_style_tl
5428     \ignorespaces
5429 }

```

## 22 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).

```
5430 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5431 {
```

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.

```
5432 \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`.

```
5433 \str_if_in:nnF { #1 } { !! }
5434 {
5435   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5436   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5437 }
5438 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5439 {
5440   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5441   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5442 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5443 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5444 }

5445 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5446 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5447 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5448 {
5449   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5450   {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5451 \group_begin:
5452 \pgfsetcornersarced
5453 {
5454   \pgfpoint
5455   { \l_@@_tab_rounded_corners_dim }
5456   { \l_@@_tab_rounded_corners_dim }
5457 }
```

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.

```

5458     \bool_if:NTF \l_@@_hvlines_bool
5459     {
5460         \pgfpathrectanglecorners
5461         {
5462             \pgfpointadd
5463             { \@@_qpoint:n { row-1 } }
5464             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5465         }
5466         {
5467             \pgfpointadd
5468             {
5469                 \@@_qpoint:n
5470                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5471             }
5472             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5473         }
5474     }
5475     {
5476         \pgfpathrectanglecorners
5477         { \@@_qpoint:n { row-1 } }
5478         {
5479             \pgfpointadd
5480             {
5481                 \@@_qpoint:n
5482                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5483             }
5484             { \pgfpoint \c_zero_dim \arrayrulewidth }
5485         }
5486     }
5487     \pgfusepath { clip }
5488     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5489     }
5490 }

```

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`).

```

5491 \cs_new_protected:Npn \@@_actually_color:
5492 {
5493     \pgfpicture
5494     \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.

```

5495     \@@_clip_with_rounded_corners:
5496     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5497     {
5498         \int_compare:nNnTF { ##1 } = \c_one_int
5499         {
5500             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5501             \use:c { g_@@_color _ 1 _tl }
5502             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5503         }
5504         {
5505             \begin { pgfscope }
5506                 \@@_color_opacity ##2
5507                 \use:c { g_@@_color _ ##1 _tl }

```

```

5508         \tl_gclear:c { g_@@_color _ ##1 _tl }
5509         \pgfusepath { fill }
5510     \end { pgfscope }
5511 }
5512 }
5513 \endpgfpicture
5514 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5515 \cs_new_protected:Npn \@@_color_opacity
5516 {
5517     \peek_meaning:NTF [
5518         { \@@_color_opacity:w }
5519         { \@@_color_opacity:w [ ] }
5520     }

```

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.

```

5521 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5522 {
5523     \tl_clear:N \l_tmpa_tl
5524     \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.

```

5525     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5526     \tl_if_empty:NTF \l_tmpb_tl
5527         { \@declaredcolor }
5528         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5529 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5530 \keys_define:nn { nicematrix / color-opacity }
5531 {
5532     opacity .tl_set:N          = \l_tmpa_tl ,
5533     opacity .value_required:n = true
5534 }

5535 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5536 {
5537     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5538     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5539     \@@_cartesian_path:
5540 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5541 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5542 {
5543     \tl_if_blank:nF { #2 }
5544     {
5545         \@@_add_to_colors_seq:en
5546         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5547         { \@@_cartesian_color:nn { #3 } { - } }
5548     }
5549 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5550 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5551 {
5552   \tl_if_blank:nF { #2 }
5553   {
5554     \@@_add_to_colors_seq:en
5555     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5556     { \@@_cartesian_color:nn { - } { #3 } }
5557   }
5558 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5559 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5560 {
5561   \tl_if_blank:nF { #2 }
5562   {
5563     \@@_add_to_colors_seq:en
5564     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5565     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5566   }
5567 }

```

The last argument is the radius of the corners of the rectangle.

```

5568 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5569 {
5570   \tl_if_blank:nF { #2 }
5571   {
5572     \@@_add_to_colors_seq:en
5573     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5574     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5575   }
5576 }

```

The last argument is the radius of the corners of the rectangle.

```

5577 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5578 {
5579   \@@_cut_on_hyphen:w #1 \q_stop
5580   \tl_clear_new:N \l_@@_tmpc_tl
5581   \tl_clear_new:N \l_@@_tmpd_tl
5582   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5583   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5584   \@@_cut_on_hyphen:w #2 \q_stop
5585   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5586   \tl_set:Nx \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`.

```

5587   \@@_cartesian_path:n { #3 }
5588 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5589 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5590 {
5591   \clist_map_inline:nn { #3 }
5592   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5593 }

```

```

5594 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5595 {
5596   \int_step_inline:nn \c@iRow

```

```

5597     {
5598       \int_step_inline:nn \c@jCol
5599       {
5600         \int_if_even:nTF { ####1 + ##1 }
5601         { \@@_cellcolor [ #1 ] { #2 } }
5602         { \@@_cellcolor [ #1 ] { #3 } }
5603         { ##1 - ####1 }
5604       }
5605     }
5606   }

```

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”.

```

5607 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5608 {
5609   \@@_rectanglecolor [ #1 ] { #2 }
5610   { 1 - 1 }
5611   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5612 }

5613 \keys_define:nn { NiceMatrix / rowcolors }
5614 {
5615   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5616   respect-blocks .default:n = true ,
5617   cols .tl_set:N = \l_@@_cols_tl ,
5618   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5619   restart .default:n = true ,
5620   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5621 }

```

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*.

```

5622 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5623 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5624   \group_begin:
5625   \seq_clear_new:N \l_@@_colors_seq
5626   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5627   \tl_clear_new:N \l_@@_cols_tl
5628   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5629   \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).

```

5630   \int_zero_new:N \l_@@_color_int
5631   \int_set_eq:NN \l_@@_color_int \c_one_int
5632   \bool_if:NT \l_@@_respect_blocks_bool
5633   {

```

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 a the sequence `\l_tmpa_seq`).

```

5634     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5635     \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5636     { \@@_not_in_exterior_p:nnnnn ##1 }
5637   }

```

```

5638 \pgfpicture
5639 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5640 \clist_map_inline:nn { #2 }
5641 {
5642   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5643   \tl_if_in:NnTF \l_tmpa_tl { - }
5644   { \@@_cut_on_hyphen:w ##1 \q_stop }
5645   { \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.

```

5646   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5647   \int_set:Nn \l_@@_color_int
5648   { \bool_if:NNTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5649   \int_zero_new:N \l_@@_tmpc_int
5650   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5651   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5652   {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5653     \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).

```

5654     \bool_if:NT \l_@@_respect_blocks_bool
5655     {
5656       \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5657       { \@@_intersect_our_row_p:nnnnn ###1 }
5658       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5659     }
5660     \tl_set:No \l_@@_rows_tl
5661     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5662     \tl_clear_new:N \l_@@_color_tl
5663     \tl_set:Nx \l_@@_color_tl
5664     {
5665       \@@_color_index:n
5666       {
5667         \int_mod:nn
5668         { \l_@@_color_int - 1 }
5669         { \seq_count:N \l_@@_colors_seq }
5670         + 1
5671       }
5672     }
5673     \tl_if_empty:NF \l_@@_color_tl
5674     {
5675       \@@_add_to_colors_seq:ee
5676       { \tl_if_blank:NF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5677       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5678     }
5679     \int_incr:N \l_@@_color_int
5680     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5681   }
5682 }
5683 \endpgfpicture
5684 \group_end:
5685 }

```

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.

```

5686 \cs_new:Npn \@@_color_index:n #1
5687 {

```

```

5688 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5689 { \@@_color_index:n { #1 - 1 } }
5690 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5691 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5692 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5693 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5694 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5695 {
5696   \int_compare:nNt { #3 } > \l_tmpb_int
5697   { \int_set:Nn \l_tmpb_int { #3 } }
5698 }

```

```

5699 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5700 {
5701   \int_if_zero:nTF { #4 }
5702   \prg_return_false:
5703   {
5704     \int_compare:nNtTF { #2 } > \c@jCol
5705     \prg_return_false:
5706     \prg_return_true:
5707   }
5708 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5709 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5710 {
5711   \int_compare:nNtTF { #1 } > \l_tmpa_int
5712   \prg_return_false:
5713   {
5714     \int_compare:nNtTF \l_tmpa_int > { #3 }
5715     \prg_return_false:
5716     \prg_return_true:
5717   }
5718 }

```

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`).

```

5719 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5720 {
5721   \dim_compare:nNtTF { #1 } = \c_zero_dim
5722   {
5723     \bool_if:NTF
5724     \@@_nocolor_used_bool
5725     \@@_cartesian_path_normal_ii:
5726     {
5727       \seq_if_empty:NTF \l_@@_corners_cells_seq
5728       { \@@_cartesian_path_normal_i:n { #1 } }
5729       \@@_cartesian_path_normal_ii:
5730     }

```

```

5731     }
5732     { \@@_cartesian_path_normal_i:n { #1 } }
5733 }

```

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.

```

5734 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5735 {
5736   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5737   \clist_map_inline:Nn \l_@@_cols_tl
5738   {
5739     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5740     \tl_if_in:NnTF \l_tmpa_tl { - }
5741     { \@@_cut_on_hyphen:w ##1 \q_stop }
5742     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5743     \tl_if_empty:NNTF \l_tmpa_tl
5744     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5745     {
5746       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5747       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5748     }
5749     \tl_if_empty:NNTF \l_tmpb_tl
5750     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5751     {
5752       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5753       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5754     }
5755     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5756     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5757     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5758     \@@_qpoint:n { col - \l_tmpa_tl }
5759     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5760     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5761     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5762     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5763     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5764   \clist_map_inline:Nn \l_@@_rows_tl
5765   {
5766     \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5767     \tl_if_in:NnTF \l_tmpa_tl { - }
5768     { \@@_cut_on_hyphen:w #####1 \q_stop }
5769     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5770     \tl_if_empty:NNTF \l_tmpa_tl
5771     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5772     {
5773       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5774       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5775     }
5776     \tl_if_empty:NNTF \l_tmpb_tl
5777     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5778     {
5779       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5780       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5781     }
5782     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5783     { \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`.

```

5784         \cs_if_exist:cF
5785         { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5786         {
5787             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5788             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5789             \@@_qpoint:n { row - \l_tmpa_tl }
5790             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5791             \pgfpathrectanglecorners
5792             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5793             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5794         }
5795     }
5796 }
5797 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5798 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5799 {
5800     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5801     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5802     \clist_map_inline:Nn \l_@@_cols_tl
5803     {
5804         \@@_qpoint:n { col - ##1 }
5805         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5806         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5807         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5808         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5809         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5810     \clist_map_inline:Nn \l_@@_rows_tl
5811     {
5812         \seq_if_in:NnF \l_@@_corners_cells_seq
5813         { #####1 - ##1 }
5814         {
5815             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5816             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5817             \@@_qpoint:n { row - #####1 }
5818             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5819             \cs_if_exist:cF
5820             { @@ _ #####1 _ ##1 _ nocolor }
5821             {
5822                 \pgfpathrectanglecorners
5823                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5824                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5825             }
5826         }
5827     }
5828 }
5829 }

```

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`).

```

5830 \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

```

5831 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1

```

```

5832 {
5833   \bool_set_true:N \@@_nocolor_used_bool
5834   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5835   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5836   \clist_map_inline:Nn \l_@@_rows_tl
5837   {
5838     \clist_map_inline:Nn \l_@@_cols_tl
5839     { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5840   }
5841 }

```

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`.

```

5842 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5843 {
5844   \clist_set_eq:NN \l_tmpa_clist #1
5845   \clist_clear:N #1
5846   \clist_map_inline:Nn \l_tmpa_clist
5847   {
5848     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5849     \tl_if_in:NnTF \l_tmpa_tl { - }
5850     { \@@_cut_on_hyphen:w ##1 \q_stop }
5851     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5852     \bool_lazy_or:nnT
5853     { \tl_if_blank_p:o \l_tmpa_tl }
5854     { \str_if_eq_p:on \l_tmpa_tl { * } }
5855     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5856     \bool_lazy_or:nnT
5857     { \tl_if_blank_p:o \l_tmpb_tl }
5858     { \str_if_eq_p:on \l_tmpb_tl { * } }
5859     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5860     \int_compare:nNnT \l_tmpb_tl > #2
5861     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5862     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5863     { \clist_put_right:Nn #1 { #####1 } }
5864   }
5865 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5866 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5867 {
5868   \@@_test_color_inside:
5869   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5870   {

```

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`).

```

5871     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5872     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5873   }
5874   \ignorespaces
5875 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5876 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5877 {
5878   \@@_test_color_inside:

```

```

5879 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5880 {
5881   \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5882   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5883   { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5884 }
5885 \ignorespaces
5886 }

```

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 currying).

```

5887 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5888 { \@@_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.

```

5889 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5890 {
5891   \@@_test_color_inside:
5892   \peek_remove_spaces:n
5893   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5894 }

```

```

5895 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5896 {

```

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`.

```

5897 \seq_gclear:N \g_tmpa_seq
5898 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5899 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5900 \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`.

```

5901 \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5902 {
5903   { \int_use:N \c@iRow }
5904   { \exp_not:n { #1 } }
5905   { \exp_not:n { #2 } }
5906   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5907 }
5908 }

```

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`).

```

5909 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5910 {
5911   \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).

```

5912     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5913     {
5914       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5915       {
5916         \@@_rowlistcolors
5917         [ \exp_not:n { #2 } ]
5918         { #1 - \int_eval:n { \c@iRow - 1 } }
5919         { \exp_not:n { #3 } }
5920         [ \exp_not:n { #4 } ]
5921       }
5922     }
5923 }

```

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.

```

5924 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5925 {
5926   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5927   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5928   \seq_gclear:N \g_@@_rowlistcolors_seq
5929 }

5930 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5931 {
5932   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5933   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5934 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5935 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5936 {

```

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).

```

5937   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5938   {

```

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).

```

5939     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5940     {
5941       \exp_not:N \columncolor [ #1 ]
5942       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5943     }
5944   }
5945 }

5946 \hook_gput_code:nnn { begindocument } { . }
5947 {
5948   \IfPackageLoadedTF { colortbl }
5949   {
5950     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5951     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5952     \cs_new_protected:Npn \@@_revert_colortbl:

```

```

5953     {
5954         \hook_gput_code:nnn { env / tabular / begin } { . }
5955         {
5956             \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5957             \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5958         }
5959     }
5960 }
5961 { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5962 }

```

## 23 The vertical and horizontal rules

### OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` 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`.

```

5963 \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`.

```

5964 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5965 {
5966     \int_if_zero:nTF \l_@@_first_col_int
5967     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5968     {
5969         \int_if_zero:nTF \c@jCol
5970         {
5971             \int_compare:nNnF \c@iRow = { -1 }
5972             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5973         }
5974         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5975     }
5976 }

```

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.

```

5977 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5978 {
5979     \int_if_zero:nF \c@iRow
5980     {
5981         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5982         {
5983             \int_compare:nNnT \c@jCol > \c_zero_int
5984             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5985         }
5986     }
5987 }

```

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.

```

5988 \keys_define:nn { NiceMatrix / Rules }
5989 {
5990   position .int_set:N = \l_@@_position_int ,
5991   position .value_required:n = true ,
5992   start .int_set:N = \l_@@_start_int ,
5993   end .code:n =
5994     \bool_lazy_or:nnTF
5995     { \tl_if_empty_p:n { #1 } }
5996     { \str_if_eq_p:nn { #1 } { last } }
5997     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5998     { \int_set:Nn \l_@@_end_int { #1 } }
5999 }

```

It’s possible that the rule won’t be drawn continuously from `start` ot `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.

```

6000 \keys_define:nn { NiceMatrix / RulesBis }
6001 {
6002   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6003   multiplicity .initial:n = 1 ,
6004   dotted .bool_set:N = \l_@@_dotted_bool ,
6005   dotted .initial:n = false ,
6006   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]`.

```

6007   color .code:n =
6008     \@@_set_CT@arc@:n { #1 }
6009     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6010   color .value_required:n = true ,
6011   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6012   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.

```

6013   tikz .code:n =
6014     \IfPackageLoadedTF { tikz }
6015     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6016     { \@@_error:n { tikz-without-tikz } } ,
6017   tikz .value_required:n = true ,
6018   total-width .dim_set:N = \l_@@_rule_width_dim ,
6019   total-width .value_required:n = true ,
6020   width .meta:n = { total-width = #1 } ,
6021   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
6022 }

```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of *key=value* pairs.

```

6023 \cs_new_protected:Npn \@@_vline:n #1
6024 {

```

The group is for the options.

```

6025 \group_begin:
6026 \int_set_eq:NN \l_@@_end_int \c@iRow
6027 \keys_set_known:nN { 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).

```

6028 \int_compare:nNt \l_@@_position_int < { \c@jCol + 2 }
6029 \@@_vline_i:
6030 \group_end:
6031 }

6032 \cs_new_protected:Npn \@@_vline_i:
6033 {

```

`\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`.

```

6034 \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6035 \int_step_variable:nNn \l_@@_start_int \l_@@_end_int
6036 \l_tmpa_tl
6037 {

```

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.

```

6038 \bool_gset_true:N \g_tmpa_bool
6039 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6040 { \@@_test_vline_in_block:nnnn ##1 }
6041 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6042 { \@@_test_vline_in_block:nnnn ##1 }
6043 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6044 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6045 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6046 \bool_if:NTF \g_tmpa_bool
6047 {
6048 \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.

```

6049 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6050 }
6051 {
6052 \int_compare:nNt \l_@@_local_start_int > \c_zero_int
6053 {
6054 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6055 \@@_vline_ii:
6056 \int_zero:N \l_@@_local_start_int
6057 }
6058 }
6059 }
6060 \int_compare:nNt \l_@@_local_start_int > \c_zero_int
6061 {
6062 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6063 \@@_vline_ii:
6064 }
6065 }

```

```

6066 \cs_new_protected:Npn \@@_test_in_corner_v:
6067 {
6068 \int_compare:nNtTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6069 {
6070 \seq_if_in:NxT
6071 \l_@@_corners_cells_seq

```

```

6072         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6073         { \bool_set_false:N \g_tmpa_bool }
6074     }
6075     {
6076         \seq_if_in:NxT
6077         \l_@@_corners_cells_seq
6078         { \l_tmpa_tl - \l_tmpb_tl }
6079         {
6080             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6081             { \bool_set_false:N \g_tmpa_bool }
6082             {
6083                 \seq_if_in:NxT
6084                 \l_@@_corners_cells_seq
6085                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6086                 { \bool_set_false:N \g_tmpa_bool }
6087             }
6088         }
6089     }
6090 }

6091 \cs_new_protected:Npn \@@_vline_ii:
6092 {
6093     \tl_clear:N \l_@@_tikz_rule_tl
6094     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6095     \bool_if:NTF \l_@@_dotted_bool
6096     \@@_vline_iv:
6097     {
6098         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6099         \@@_vline_iii:
6100         \@@_vline_v:
6101     }
6102 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6103 \cs_new_protected:Npn \@@_vline_iii:
6104 {
6105     \pgfpicture
6106     \pgfrememberpicturepositiononpagetrue
6107     \pgf@relevantforpicturesizefalse
6108     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6109     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6110     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6111     \dim_set:Nn \l_tmpb_dim
6112     {
6113         \pgf@x
6114         - 0.5 \l_@@_rule_width_dim
6115         +
6116         ( \arrayrulewidth * \l_@@_multiplicity_int
6117           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6118     }
6119     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6120     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6121     \bool_lazy_all:nT
6122     {
6123         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6124         { \cs_if_exist_p:N \CT@drsc@ }
6125         { ! \tl_if_blank_p:o \CT@drsc@ }
6126     }
6127     {
6128         \group_begin:
6129         \CT@drsc@
6130         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

```

6131 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6132 \dim_set:Nn \l_@@_tmpd_dim
6133 {
6134 \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6135 * ( \l_@@_multiplicity_int - 1 )
6136 }
6137 \pgfpathrectanglecorners
6138 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6139 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6140 \pgfusepath { fill }
6141 \group_end:
6142 }
6143 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6144 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6145 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6146 {
6147 \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6148 \dim_sub:Nn \l_tmpb_dim \doublerulesep
6149 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6150 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6151 }
6152 \CT@arc@
6153 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6154 \pgfsetrectcap
6155 \pgfusepathqstroke
6156 \endpgfpicture
6157 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6158 \cs_new_protected:Npn \@@_vline_iv:
6159 {
6160 \pgfpicture
6161 \pgfrememberpicturepositiononpagetrue
6162 \pgf@relevantforpicturesizefalse
6163 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6164 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6165 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6166 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6167 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6168 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6169 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6170 \CT@arc@
6171 \@@_draw_line:
6172 \endpgfpicture
6173 }

```

The following code is for the case when the user uses the key `tikz`.

```

6174 \cs_new_protected:Npn \@@_vline_v:
6175 {
6176 \begin {tikzpicture }
6177 % added 2023/09/25

```

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).

```

6178 \CT@arc@
6179 \tl_if_empty:NF \l_@@_rule_color_tl
6180 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6181 \pgfrememberpicturepositiononpagetrue
6182 \pgf@relevantforpicturesizefalse
6183 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6184 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6185 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }

```

```

6186 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6187 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6188 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6189 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6190 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6191 ( \l_tmpb_dim , \l_tmpa_dim ) --
6192 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6193 \end { tikzpicture }
6194 }

```

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).

```

6195 \cs_new_protected:Npn \@@_draw_vlines:
6196 {
6197   \int_step_inline:nnn
6198   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6199   {
6200     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6201     \c@jCol
6202     { \int_eval:n { \c@jCol + 1 } } }
6203   }
6204   {
6205     \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6206     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6207     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6208   }
6209 }

```

## 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}`.

```

6210 \cs_new_protected:Npn \@@_hline:n #1
6211 {

```

The group is for the options.

```

6212   \group_begin:
6213   \int_zero_new:N \l_@@_end_int
6214   \int_set_eq:NN \l_@@_end_int \c@jCol
6215   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6216   \@@_hline_i:
6217   \group_end:
6218 }

```

```

6219 \cs_new_protected:Npn \@@_hline_i:
6220 {
6221   \int_zero_new:N \l_@@_local_start_int
6222   \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`.

```

6223   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6224   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6225   \l_tmpb_tl
6226   {

```

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.

```

6227   \bool_gset_true:N \g_tmpa_bool
6228   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6229   { \@@_test_hline_in_block:nnnnn ##1 }

```

```

6230 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6231 { \@@_test_hline_in_block:nnnnn ##1 }
6232 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6233 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6234 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6235 \bool_if:NTF \g_tmpa_bool
6236 {
6237   \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.

```

6238     { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6239   }
6240   {
6241     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6242     {
6243       \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6244       \@@_hline_ii:
6245       \int_zero:N \l_@@_local_start_int
6246     }
6247   }
6248 }
6249 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6250 {
6251   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6252   \@@_hline_ii:
6253 }
6254 }

```

```

6255 \cs_new_protected:Npn \@@_test_in_corner_h:
6256 {
6257   \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6258   {
6259     \seq_if_in:NxT
6260     \l_@@_corners_cells_seq
6261     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6262     { \bool_set_false:N \g_tmpa_bool }
6263   }
6264   {
6265     \seq_if_in:NxT
6266     \l_@@_corners_cells_seq
6267     { \l_tmpa_tl - \l_tmpb_tl }
6268     {
6269       \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6270       { \bool_set_false:N \g_tmpa_bool }
6271       {
6272         \seq_if_in:NxT
6273         \l_@@_corners_cells_seq
6274         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6275         { \bool_set_false:N \g_tmpa_bool }
6276       }
6277     }
6278   }
6279 }

```

```

6280 \cs_new_protected:Npn \@@_hline_ii:
6281 {
6282   \tl_clear:N \l_@@_tikz_rule_tl
6283   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6284   \bool_if:NTF \l_@@_dotted_bool
6285     \@@_hline_iv:
6286   {

```

```

6287     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6288     \@@_hline_iii:
6289     \@@_hline_v:
6290   }
6291 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6292 \cs_new_protected:Npn \@@_hline_iii:
6293 {
6294   \pgfpicture
6295   \pgfrememberpicturepositiononpagetrue
6296   \pgf@relevantforpicturesizefalse
6297   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6298   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6299   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6300   \dim_set:Nn \l_tmpb_dim
6301   {
6302     \pgf@y
6303     - 0.5 \l_@@_rule_width_dim
6304     +
6305     ( \arrayrulewidth * \l_@@_multiplicity_int
6306       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6307   }
6308   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6309   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6310   \bool_lazy_all:nT
6311   {
6312     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6313     { \cs_if_exist_p:N \CT@drsc@ }
6314     { ! \tl_if_blank_p:o \CT@drsc@ }
6315   }
6316   {
6317     \group_begin:
6318     \CT@drsc@
6319     \dim_set:Nn \l_@@_tmpd_dim
6320     {
6321       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6322       * ( \l_@@_multiplicity_int - 1 )
6323     }
6324     \pgfpathrectanglecorners
6325     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6326     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6327     \pgfusepathqfill
6328     \group_end:
6329   }
6330   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6331   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6332   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6333   {
6334     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6335     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6336     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6337     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6338   }
6339   \CT@arc@
6340   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6341   \pgfsetrectcap
6342   \pgfusepathqstroke
6343   \endpgfpicture
6344 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6345 \cs_new_protected:Npn \@@_hline_iv:
6346 {
6347   \pgfpicture
6348   \pgfrememberpicturepositiononpagetrue
6349   \pgf@relevantforpicturesizefalse
6350   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6351   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6352   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6353   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6354   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6355   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6356   {
6357     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6358     \bool_if:NF \g_@@_delims_bool
6359     { \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.

```

6360     \tl_if_eq:NnF \g_@@_left_delim_tl (
6361       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6362     )
6363     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6364     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6365     \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6366     {
6367       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6368       \bool_if:NF \g_@@_delims_bool
6369       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6370       \tl_if_eq:NnF \g_@@_right_delim_tl (
6371         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6372       )
6373     }
6374     \CT@arc@
6375     \@@_draw_line:
6376     \endpgfpicture

```

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`).

```

6377 \cs_new_protected:Npn \@@_hline_v:
6378 {
6379   \begin { tikzpicture }
6380   % added 2023/09/25

```

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).

```

6381   \CT@arc@

```

```

6382 \tl_if_empty:NF \l_@@_rule_color_tl
6383 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6384 \pgfrememberpicturepositiononpagetrue
6385 \pgf@relevantforpicturesizefalse
6386 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6387 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6388 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6389 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6390 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6391 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6392 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6393 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6394 ( \l_tmpa_dim , \l_tmpb_dim ) --
6395 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6396 \end { tikzpicture }
6397 }

```

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).

```

6398 \cs_new_protected:Npn \@@_draw_hlines:
6399 {
6400   \int_step_inline:nnn
6401   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6402   {
6403     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6404     \c@iRow
6405     { \int_eval:n { \c@iRow + 1 } }
6406   }
6407   {
6408     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6409     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6410     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6411   }
6412 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6413 \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.

```

6414 \cs_set:Npn \@@_Hline_i:n #1
6415 {
6416   \peek_remove_spaces:n
6417   {
6418     \peek_meaning:NTF \Hline
6419     { \@@_Hline_ii:nn { #1 + 1 } }
6420     { \@@_Hline_iii:n { #1 } }
6421   }
6422 }
6423 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6424 \cs_set:Npn \@@_Hline_iii:n #1
6425 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6426 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6427 {
6428   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6429   \skip_vertical:N \l_@@_rule_width_dim
6430   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6431   {
6432     \@@_hline:n
6433     {
6434       multiplicity = #1 ,

```

```

6435         position = \int_eval:n { \c@iRow + 1 } ,
6436         total-width = \dim_use:N \l_@@_rule_width_dim ,
6437         #2
6438     }
6439 }
6440 \egroup
6441 }

```

### 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`).

```

6442 \cs_new_protected:Npn \@@_custom_line:n #1
6443 {
6444     \str_clear_new:N \l_@@_command_str
6445     \str_clear_new:N \l_@@_ccommand_str
6446     \str_clear_new:N \l_@@_letter_str
6447     \tl_clear_new:N \l_@@_other_keys_tl
6448     \keys_set_known:nnN { 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.

```

6449     \bool_lazy_all:nTF
6450     {
6451         { \str_if_empty_p:N \l_@@_letter_str }
6452         { \str_if_empty_p:N \l_@@_command_str }
6453         { \str_if_empty_p:N \l_@@_ccommand_str }
6454     }
6455     { \@@_error:n { No~letter~and~no~command } }
6456     { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6457 }
6458 \keys_define:nn { NiceMatrix / custom-line }
6459 {
6460     letter .str_set:N = \l_@@_letter_str ,
6461     letter .value_required:n = true ,
6462     command .str_set:N = \l_@@_command_str ,
6463     command .value_required:n = true ,
6464     ccommand .str_set:N = \l_@@_ccommand_str ,
6465     ccommand .value_required:n = true ,
6466 }

```

```

6467 \cs_new_protected:Npn \@@_custom_line_i:n #1
6468 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6469     \bool_set_false:N \l_@@_tikz_rule_bool
6470     \bool_set_false:N \l_@@_dotted_rule_bool
6471     \bool_set_false:N \l_@@_color_bool
6472     \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6473     \bool_if:NT \l_@@_tikz_rule_bool
6474     {
6475         \IfPackageLoadedTF { tikz }
6476         { }
6477         { \@@_error:n { tikz~in~custom~line~without~tikz } }
6478     }
6479     \bool_if:NT \l_@@_color_bool
6480     { \@@_error:n { color~in~custom~line~with~tikz } }

```

```

6480     }
6481     \bool_if:NT \l_@@_dotted_rule_bool
6482     {
6483         \int_compare:nNtT \l_@@_multiplicity_int > \c_one_int
6484         { \@@_error:n { key~multiplicity~with~dotted } }
6485     }
6486     \str_if_empty:NF \l_@@_letter_str
6487     {
6488         \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6489         { \@@_error:n { Several~letters } }
6490         {
6491             \exp_args:NnV \tl_if_in:NnTF
6492             \c_@@_forbidden_letters_str \l_@@_letter_str
6493             { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6494         }

```

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.

```

6495         \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6496         { \@@_v_custom_line:n { #1 } }
6497     }
6498 }
6499 }
6500 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6501 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6502 }
6503 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6504 \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.

```

6505 \keys_define:nn { NiceMatrix / custom-line-bis }
6506 {
6507     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6508     multiplicity .initial:n = 1 ,
6509     multiplicity .value_required:n = true ,
6510     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6511     color .value_required:n = true ,
6512     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6513     tikz .value_required:n = true ,
6514     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6515     dotted .value_forbidden:n = true ,
6516     total-width .code:n = { } ,
6517     total-width .value_required:n = true ,
6518     width .code:n = { } ,
6519     width .value_required:n = true ,
6520     sep-color .code:n = { } ,
6521     sep-color .value_required:n = true ,
6522     unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6523 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6524 \bool_new:N \l_@@_dotted_rule_bool
6525 \bool_new:N \l_@@_tikz_rule_bool
6526 \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`.

```

6527 \keys_define:nn { NiceMatrix / custom-line-width }
6528 {
6529     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6530     multiplicity .initial:n = 1 ,
6531     multiplicity .value_required:n = true ,
6532     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6533     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6534         \bool_set_true:N \l_@@_total_width_bool ,
6535     total-width .value_required:n = true ,
6536     width .meta:n = { total-width = #1 } ,
6537     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6538 }

```

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).

```

6539 \cs_new_protected:Npn \@@_h_custom_line:n #1
6540 {

```

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).

```

6541     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6542     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6543 }

```

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).

```

6544 \cs_new_protected:Npn \@@_c_custom_line:n #1
6545 {

```

Here, we need an expandable command since it begins with an \noalign.

```

6546     \exp_args:Nc \NewExpandableDocumentCommand
6547     { nicematrix - \l_@@_ccommand_str }
6548     { 0 { } m }
6549     {
6550         \noalign
6551         {
6552             \@@_compute_rule_width:n { #1 , ##1 }
6553             \skip_vertical:n { \l_@@_rule_width_dim }
6554             \clist_map_inline:nn
6555             { ##2 }
6556             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6557         }
6558     }
6559     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6560 }

```

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*.

```

6561 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6562 {
6563     \str_if_in:nnTF { #2 } { - }
6564     { \@@_cut_on_hyphen:w #2 \q_stop }
6565     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6566     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6567     {
6568         \@@_hline:n
6569         {
6570             #1 ,
6571             start = \l_tmpa_tl ,

```

```

6572         end = \l_tmpb_tl ,
6573         position = \int_eval:n { \c@iRow + 1 } ,
6574         total-width = \dim_use:N \l_@@_rule_width_dim
6575     }
6576 }
6577 }
6578 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6579 {
6580     \bool_set_false:N \l_@@_tikz_rule_bool
6581     \bool_set_false:N \l_@@_total_width_bool
6582     \bool_set_false:N \l_@@_dotted_rule_bool
6583     \keys_set_known:n { NiceMatrix / custom-line-width } { #1 }
6584     \bool_if:NF \l_@@_total_width_bool
6585     {
6586         \bool_if:NTF \l_@@_dotted_rule_bool
6587         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6588         {
6589             \bool_if:NF \l_@@_tikz_rule_bool
6590             {
6591                 \dim_set:Nn \l_@@_rule_width_dim
6592                 {
6593                     \arrayrulewidth * \l_@@_multiplicity_int
6594                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6595                 }
6596             }
6597         }
6598     }
6599 }
6600 \cs_new_protected:Npn \@@_v_custom_line:n #1
6601 {
6602     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6603     \tl_gput_right:Nx \g_@@_array_preamble_tl
6604     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6605     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6606     {
6607         \@@_vline:n
6608         {
6609             #1 ,
6610             position = \int_eval:n { \c@jCol + 1 } ,
6611             total-width = \dim_use:N \l_@@_rule_width_dim
6612         }
6613     }
6614     \@@_rec_preamble:n
6615 }
6616 \@@_custom_line:n
6617 { 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.

```

6618 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6619 {
6620     \int_compare:nNt \l_tmpa_tl > { #1 }
6621     {
6622         \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
6623         {
6624             \int_compare:nNt \l_tmpb_tl > { #2 - 1 }

```

```

6625         {
6626         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6627         { \bool_gset_false:N \g_tmpa_bool }
6628         }
6629     }
6630 }
6631 }

```

The same for vertical rules.

```

6632 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6633 {
6634     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6635     {
6636         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6637         {
6638             \int_compare:nNnT \l_tmpb_tl > { #2 }
6639             {
6640                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6641                 { \bool_gset_false:N \g_tmpa_bool }
6642             }
6643         }
6644     }
6645 }
6646 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6647 {
6648     \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6649     {
6650         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6651         {
6652             \int_compare:nNnTF \l_tmpa_tl = { #1 }
6653             { \bool_gset_false:N \g_tmpa_bool }
6654             {
6655                 \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6656                 { \bool_gset_false:N \g_tmpa_bool }
6657             }
6658         }
6659     }
6660 }
6661 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6662 {
6663     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6664     {
6665         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6666         {
6667             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6668             { \bool_gset_false:N \g_tmpa_bool }
6669             {
6670                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6671                 { \bool_gset_false:N \g_tmpa_bool }
6672             }
6673         }
6674     }
6675 }

```

## 24 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.

```

6676 \cs_new_protected:Npn \@@_compute_corners:
6677 {

```

The sequence `\l_@@_corners_cells_seq` will be the sequence of all the empty cells (and not in a block) considered in the corners of the array.

```

6678   \seq_clear_new:N \l_@@_corners_cells_seq
6679   \clist_map_inline:Nn \l_@@_corners_clist
6680   {
6681     \str_case:nnF { ##1 }
6682     {
6683       { NW }
6684       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6685       { NE }
6686       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6687       { SW }
6688       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6689       { SE }
6690       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6691     }
6692     { \@@_error:nn { bad~corner } { ##1 } }
6693   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6694   \seq_if_empty:NF \l_@@_corners_cells_seq
6695   {

```

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 color the `rows`, `columns` and `cells` must not color the cells in the corners.

```

6696     \tl_gput_right:Nx \g_@@_aux_tl
6697     {
6698       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6699       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6700     }
6701   }
6702 }

```

“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_seq`.

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.

```

6703 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6704 {

```

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.

```

6705   \bool_set_false:N \l_tmpa_bool
6706   \int_zero_new:N \l_@@_last_empty_row_int
6707   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6708   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6709   {
6710     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6711     \bool_lazy_or:nnTF

```

```

6712     {
6713         \cs_if_exist_p:c
6714         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6715     }
6716     \l_tmpb_bool
6717     { \bool_set_true:N \l_tmpa_bool }
6718     {
6719         \bool_if:NF \l_tmpa_bool
6720         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6721     }
6722 }

```

Now, you determine the last empty cell in the row of number 1.

```

6723     \bool_set_false:N \l_tmpa_bool
6724     \int_zero_new:N \l_@@_last_empty_column_int
6725     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6726     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6727     {
6728         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6729         \bool_lazy_or:nnTF
6730         \l_tmpb_bool
6731         {
6732             \cs_if_exist_p:c
6733             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6734         }
6735         { \bool_set_true:N \l_tmpa_bool }
6736         {
6737             \bool_if:NF \l_tmpa_bool
6738             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6739         }
6740     }

```

Now, we loop over the rows.

```

6741     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6742     {

```

We treat the row number ##1 with another loop.

```

6743         \bool_set_false:N \l_tmpa_bool
6744         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6745         {
6746             \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
6747             \bool_lazy_or:nnTF
6748             \l_tmpb_bool
6749             {
6750                 \cs_if_exist_p:c
6751                 { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
6752             }
6753             { \bool_set_true:N \l_tmpa_bool }
6754             {
6755                 \bool_if:NF \l_tmpa_bool
6756                 {
6757                     \int_set:Nn \l_@@_last_empty_column_int { ####1 }
6758                     \seq_put_right:Nn
6759                     \l_@@_corners_cells_seq
6760                     { ##1 - ####1 }
6761                 }
6762             }
6763         }
6764     }
6765 }

```

The following macro tests whether a cell is in (at least) one of the blocks of the array (or in a cell with a `\diagbox`).

The flag `\l_tmpb_bool` will be raised if the cell #1-#2 is in a block (or in a cell with a `\diagbox`).

```

6766 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6767 {
6768   \int_set:Nn \l_tmpa_int { #1 }
6769   \int_set:Nn \l_tmpb_int { #2 }
6770   \bool_set_false:N \l_tmpb_bool
6771   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6772     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6773 }

6774 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6775 {
6776   \int_compare:nNnF { #3 } > { #1 }
6777   {
6778     \int_compare:nNnF { #1 } > { #5 }
6779     {
6780       \int_compare:nNnF { #4 } > { #2 }
6781       {
6782         \int_compare:nNnF { #2 } > { #6 }
6783         { \bool_set_true:N \l_tmpb_bool }
6784       }
6785     }
6786   }
6787 }

```

## 25 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.

```

6788 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6789 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6790 {
6791   auto-columns-width .code:n =
6792   {
6793     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6794     \dim_gzero_new:N \g_@@_max_cell_width_dim
6795     \bool_set_true:N \l_@@_auto_columns_width_bool
6796   }
6797 }

6798 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6799 {
6800   \int_gincr:N \g_@@_NiceMatrixBlock_int
6801   \dim_zero:N \l_@@_columns_width_dim
6802   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6803   \bool_if:NT \l_@@_block_auto_columns_width_bool
6804   {
6805     \cs_if_exist:cT
6806       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6807     {
6808       % is \exp_args:NNe mandatory?
6809       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6810         {
6811           \use:c
6812             { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6813         }
6814     }

```

```

6815     }
6816 }

```

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`).

```

6817 {
6818   \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.

```

6819     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6820     {
6821       \bool_if:NT \l_@@_block_auto_columns_width_bool
6822       {
6823         \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6824         \iow_shipout:Nx \@mainaux
6825         {
6826           \cs_gset:cpn
6827             { @@ _ 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.

```

6828         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6829       }
6830       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6831     }
6832   }
6833   \ignorespacesafterend
6834 }

```

## 26 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

6835 \cs_generate_variant:Nn \dim_min:nn { v n }
6836 \cs_generate_variant:Nn \dim_max:nn { v n }

```

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).

```

6837 \cs_new_protected:Npn \@@_create_extra_nodes:
6838 {
6839   \bool_if:nTF \l_@@_medium_nodes_bool
6840   {
6841     \bool_if:NTF \l_@@_large_nodes_bool
6842       \@@_create_medium_and_large_nodes:
6843       \@@_create_medium_nodes:
6844   }
6845   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6846 }

```

We have three macros of creation of nodes: `\@@_create_medium_nodes:`, `\@@_create_large_nodes:` and `\@@_create_medium_and_large_nodes:`.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command `\@@_computations_for_medium_nodes:` to do these computations.

The command `\@@_computations_for_medium_nodes:` must be used in a `{pgfpicture}`.

For each row  $i$ , we compute two dimensions  $l\_@@\_row\_i\_min\_dim$  and  $l\_@@\_row\_i\_max\_dim$ . The dimension  $l\_@@\_row\_i\_min\_dim$  is the minimal  $y$ -value of all the cells of the row  $i$ . The dimension  $l\_@@\_row\_i\_max\_dim$  is the maximal  $y$ -value of all the cells of the row  $i$ .

Similarly, for each column  $j$ , we compute two dimensions  $l\_@@\_column\_j\_min\_dim$  and  $l\_@@\_column\_j\_max\_dim$ . The dimension  $l\_@@\_column\_j\_min\_dim$  is the minimal  $x$ -value of all the cells of the column  $j$ . The dimension  $l\_@@\_column\_j\_max\_dim$  is the maximal  $x$ -value of all the cells of the column  $j$ .

Since these dimensions will be computed as maximum or minimum, we initialize them to  $\backslash c\_max\_dim$  or  $-\backslash c\_max\_dim$ .

```

6847 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6848 {
6849   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6850   {
6851     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6852     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6853     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6854     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6855   }
6856   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6857   {
6858     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6859     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6860     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6861     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6862   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6863   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6864   {
6865     \int_step_variable:nnNn
6866     \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.

```

6867     {
6868       \cs_if_exist:cT
6869       { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell  $(i-j)$ . They will be stored in  $\backslash pgf@x$  and  $\backslash pgf@y$ .

```

6870     {
6871       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6872       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6873       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6874       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6875       {
6876         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6877         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6878       }

```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell  $(i-j)$ . They will be stored in  $\backslash pgf@x$  and  $\backslash pgf@y$ .

```

6879       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6880       \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6881       { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6882       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6883       {
6884         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6885         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6886       }
6887     }
6888   }
6889 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6890 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6891 {
6892   \dim_compare:nnNt
6893   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6894   {
6895     \@@_qpoint:n { row - \@@_i: - base }
6896     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6897     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6898   }
6899 }
6900 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6901 {
6902   \dim_compare:nnNt
6903   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6904   {
6905     \@@_qpoint:n { col - \@@_j: }
6906     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6907     \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6908   }
6909 }
6910 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6911 \cs_new_protected:Npn \@@_create_medium_nodes:
6912 {
6913   \pgfpicture
6914   \pgfrememberpicturepositiononpagetrue
6915   \pgf@relevantforpicturesizefalse
6916   \@@_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”.

```

6917   \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6918   \@@_create_nodes:
6919   \endpgfpicture
6920 }

```

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:`.

```

6921 \cs_new_protected:Npn \@@_create_large_nodes:
6922 {
6923   \pgfpicture
6924   \pgfrememberpicturepositiononpagetrue
6925   \pgf@relevantforpicturesizefalse
6926   \@@_computations_for_medium_nodes:
6927   \@@_computations_for_large_nodes:
6928   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6929   \@@_create_nodes:
6930   \endpgfpicture
6931 }
6932 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6933 {

```

---

<sup>14</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6934 \pgfpicture
6935 \pgfrememberpicturepositiononpagetrue
6936 \pgf@relevantforpicturesizefalse
6937 \@@_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”.

```

6938 \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6939 \@@_create_nodes:
6940 \@@_computations_for_large_nodes:
6941 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6942 \@@_create_nodes:
6943 \endpgfpicture
6944 }

```

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.

```

6945 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6946 {
6947 \int_set_eq:NN \l_@@_first_row_int \c_one_int
6948 \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`.

```

6949 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6950 {
6951 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6952 {
6953 (
6954 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6955 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6956 )
6957 / 2
6958 }
6959 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6960 { l_@@_row _ \@@_i: _ min _ dim }
6961 }
6962 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6963 {
6964 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6965 {
6966 (
6967 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6968 \dim_use:c
6969 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6970 )
6971 / 2
6972 }
6973 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6974 { l_@@_column _ \@@_j: _ max _ dim }
6975 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6976 \dim_sub:cn
6977 { l_@@_column _ 1 _ min _ dim }
6978 \l_@@_left_margin_dim
6979 \dim_add:cn
6980 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6981 \l_@@_right_margin_dim
6982 }

```

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).

```

6983 \cs_new_protected:Npn \@@_create_nodes:
6984 {
6985   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6986   {
6987     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6988     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6989       \@@_pgf_rect_node:nnnnn
6990       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6991       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6992       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6993       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6994       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6995       \str_if_empty:NF \l_@@_name_str
6996       {
6997         \pgfnodealias
6998         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6999         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7000       }
7001     }
7002   }

```

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  $n$ .

```

7003   \seq_map_pairwise_function:NNN
7004   \g_@@_multicolumn_cells_seq
7005   \g_@@_multicolumn_sizes_seq
7006   \@@_node_for_multicolumn:nn
7007 }

```

```

7008 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7009 {
7010   \cs_set_nopar:Npn \@@_i: { #1 }
7011   \cs_set_nopar:Npn \@@_j: { #2 }
7012 }

```

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”).

```

7013 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7014 {
7015   \@@_extract_coords_values: #1 \q_stop
7016   \@@_pgf_rect_node:nnnnn
7017   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7018   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7019   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7020   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: + #2 - 1 } _max_dim } }
7021   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7022   \str_if_empty:NF \l_@@_name_str
7023   {
7024     \pgfnodealias
7025     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7026     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7027   }
7028 }

```

## 27 The blocks

The 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.

```

7029 \keys_define:n { NiceMatrix / Block / FirstPass }
7030 {
7031   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7032   l .value_forbidden:n = true ,
7033   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7034   r .value_forbidden:n = true ,
7035   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7036   c .value_forbidden:n = true ,
7037   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7038   L .value_forbidden:n = true ,
7039   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7040   R .value_forbidden:n = true ,
7041   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7042   C .value_forbidden:n = true ,
7043   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7044   t .value_forbidden:n = true ,
7045   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7046   T .value_forbidden:n = true ,
7047   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7048   b .value_forbidden:n = true ,
7049   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7050   B .value_forbidden:n = true ,
7051   color .code:n =
7052     \@@_color:n { #1 }
7053     \tl_set_rescan:Nnn
7054       \l_@@_draw_tl
7055       { \char_set_catcode_other:N ! }
7056       { #1 } ,
7057   color .value_required:n = true ,
7058   respect-arraystretch .code:n =
7059     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7060   respect-arraystretch .value_forbidden:n = true ,
7061 }

```

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.

```

7062 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7063 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7064 {

```

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).

```

7065   \peek_remove_spaces:n
7066   {
7067     \tl_if_blank:nTF { #2 }
7068     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7069     {
7070       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7071       \@@_Block_i_czech \@@_Block_i
7072       #2 \q_stop
7073     }
7074     { #1 } { #3 } { #4 }

```

```

7075     }
7076 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7077 \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`.

```

7078 {
7079   \char_set_catcode_active:N -
7080   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7081 }

```

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.

```

7082 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7083 {

```

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).

```

7084   \bool_lazy_or:nnTF
7085     { \tl_if_blank_p:n { #1 } }
7086     { \str_if_eq_p:nn { #1 } { * } }
7087     { \int_set:Nn \l_tmpa_int { 100 } }
7088     { \int_set:Nn \l_tmpa_int { #1 } }
7089   \bool_lazy_or:nnTF
7090     { \tl_if_blank_p:n { #2 } }
7091     { \str_if_eq_p:nn { #2 } { * } }
7092     { \int_set:Nn \l_tmpb_int { 100 } }
7093     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7094   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7095   {
7096     \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7097       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7098       { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7099   }
7100   { \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.

```

7101   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
7102   \tl_set:Nx \l_tmpa_tl
7103   {
7104     { \int_use:N \c@iRow }
7105     { \int_use:N \c@jCol }
7106     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7107     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7108   }

```

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}`.

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by currying).

```

7109   \bool_if:nTF
7110   {
7111     (
7112       \int_compare_p:nNn \l_tmpa_int = \c_one_int
7113       ||
7114       \int_compare_p:nNn \l_tmpb_int = \c_one_int
7115     )
7116     && ! \tl_if_empty_p:n { #5 }

```

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.

```

7117     && ! \l_@@_X_bool
7118   }
7119   { \exp_args:Nee \@@_Block_iv:nnnnn }
7120   { \exp_args:Nee \@@_Block_v:nnnnn }
7121   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7122 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both). 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.

```

7123 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7124 {
7125   \int_gincr:N \g_@@_block_box_int
7126   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7127   {
7128     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7129     {
7130       \@@_actually_diagbox:nnnnnn
7131       { \int_use:N \c@iRow }
7132       { \int_use:N \c@jCol }
7133       { \int_eval:n { \c@iRow + #1 - 1 } }
7134       { \int_eval:n { \c@jCol + #2 - 1 } }
7135       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7136       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7137     }
7138   }
7139   \box_gclear_new:c
7140   { 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!

```

7141   \hbox_gset:cn
7142   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7143   {

```

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}`).

```

7144 \tl_if_empty:NTF \l_@@_color_tl
7145 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7146 { \@@_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`.

```

7147 \int_compare:nNnT { #1 } = \c_one_int
7148 {
7149   \int_if_zero:nTF \c@iRow
7150   \l_@@_code_for_first_row_tl
7151   {
7152     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7153     \l_@@_code_for_last_row_tl
7154   }
7155   \g_@@_row_style_tl
7156 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7157 \@@_reset_arraystretch:
7158 \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7159 #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.).

```

7160 \@@_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}`.

```

7161 \bool_if:NTF \l_@@_tabular_bool
7162 {
7163   \bool_lazy_all:nTF
7164   {
7165     { \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.

```

7166 { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7167 { ! \g_@@_rotate_bool }
7168 }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7169 {
7170   \use:e
7171   {
7172     \exp_not:N \begin { minipage }%
7173     [ \str_lowercase:V \l_@@_vpos_block_str ]
7174     { \l_@@_col_width_dim }
7175     \str_case:on \l_@@_hpos_block_str
7176     { c \centering r \raggedleft l \raggedright }
7177   }
7178   #5
7179   \end { minipage }
7180 }

```

In the other cases, we use a `{tabular}`.

```

7181         {
7182             \use:e
7183             {
7184                 \exp_not:N \begin { tabular }%
7185                 [ \str_lowercase:V \l_@@_vpos_block_str ]
7186                 { @ { } \l_@@_hpos_block_str @ { } }
7187             }
7188             #5
7189             \end { tabular }
7190         }
7191     }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7192     {
7193         \c_math_toggle_token
7194         \use:e
7195         {
7196             \exp_not:N \begin { array }%
7197             [ \str_lowercase:V \l_@@_vpos_block_str ]
7198             { @ { } \l_@@_hpos_block_str @ { } }
7199         }
7200         #5
7201         \end { array }
7202         \c_math_toggle_token
7203     }
7204 }

```

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 the rotated box).

```

7205     \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.

```

7206     \int_compare:nNnT { #2 } = \c_one_int
7207     {
7208         \dim_gset:Nn \g_@@_blocks_wd_dim
7209         {
7210             \dim_max:nn
7211             \g_@@_blocks_wd_dim
7212             {
7213                 \box_wd:c
7214                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7215             }
7216         }
7217     }

```

If we are in a mono-row block and if that block has no vertical option for the position<sup>15</sup>, we take into account the height and the depth of that block for the height and the depth of the row.

```

7218     \str_if_eq:VnT \l_@@_vpos_block_str { c }
7219     {
7220         \int_compare:nNnT { #1 } = \c_one_int
7221         {
7222             \dim_gset:Nn \g_@@_blocks_ht_dim
7223             {
7224                 \dim_max:nn
7225                 \g_@@_blocks_ht_dim

```

---

<sup>15</sup>If the block has a key of a vertical position, that means that it has to be put in a vertical space determined by the *others* cells of the row. Therefore there is no point creating space here. Moreover, that would lead to problems when a multi-row block with a position key such as `b` or `B`.

```

7226         {
7227             \box_ht:c
7228             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7229         }
7230     }
7231     \dim_gset:Nn \g_@@_blocks_dp_dim
7232     {
7233         \dim_max:nn
7234         \g_@@_blocks_dp_dim
7235         {
7236             \box_dp:c
7237             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7238         }
7239     }
7240 }
7241 }
7242 \seq_gput_right:Nx \g_@@_blocks_seq
7243 {
7244     \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.

```

7245     {
7246         \exp_not:n { #3 } ,
7247         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7248         \bool_if:NT \g_@@_rotate_bool
7249         {
7250             \bool_if:NTF \g_@@_rotate_c_bool
7251             { v-center }
7252             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7253         }
7254     }
7255     {
7256         \box_use_drop:c
7257         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7258     }
7259 }
7260 }
7261 \bool_set_false:N \g_@@_rotate_c_bool
7262 }

7263 \cs_new:Npn \@@_adjust_hpos_rotate:
7264 {
7265     \bool_if:NT \g_@@_rotate_bool
7266     {
7267         \str_set:Nx \l_@@_hpos_block_str
7268         {
7269             \bool_if:NTF \g_@@_rotate_c_bool
7270             { c }
7271             {
7272                 \str_case:onF \l_@@_vpos_block_str
7273                 { b l B l t r T r }
7274                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7275             }
7276         }
7277     }
7278 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7279 \cs_new_protected:Npn \@@_rotate_box_of_block:
7280 {
7281   \box_grotate:cn
7282   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7283   { 90 }
7284   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7285   {
7286     \vbox_gset_top:cn
7287     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7288     {
7289       \skip_vertical:n { 0.8 ex }
7290       \box_use:c
7291       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7292     }
7293   }
7294   \bool_if:NT \g_@@_rotate_c_bool
7295   {
7296     \hbox_gset:cn
7297     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7298     {
7299       \c_math_toggle_token
7300       \vcenter
7301       {
7302         \box_use:c
7303         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7304       }
7305       \c_math_toggle_token
7306     }
7307   }
7308 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column. In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#1 is  $i$  (the number of rows of the block), #2 is  $j$  (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7309 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7310 {
7311   \seq_gput_right:Nx \g_@@_blocks_seq
7312   {
7313     \l_tmpa_tl
7314     { \exp_not:n { #3 } }
7315     {
7316       \bool_if:NTF \l_@@_tabular_bool
7317       {
7318         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7319   \@@_reset_arraystretch:
7320   \exp_not:n
7321   {
7322     \dim_zero:N \extrarowheight
7323     #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).

```

7324         \use:e
7325         {
7326             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7327             { @ { } \l_@@_hpos_block_str @ { } }
7328         }
7329         #5
7330         \end { tabular }
7331     }
7332     \group_end:
7333 }

```

When we are *not* in an environments {NiceTabular} (or similar).

```

7334     {
7335         \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7336         \@@_reset_arraystretch:
7337         \exp_not:n
7338         {
7339             \dim_zero:N \extrarowheight
7340             #4
7341             \c_math_toggle_token
7342             \use:e
7343             {
7344                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7345                 { @ { } \l_@@_hpos_block_str @ { } }
7346             }
7347             #5
7348             \end { array }
7349             \c_math_toggle_token
7350         }
7351         \group_end:
7352     }
7353 }
7354 }
7355 }

```

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).

```

7356 \keys_define:nn { NiceMatrix / Block / SecondPass }
7357 {
7358     tikz .code:n =
7359         \IfPackageLoadedTF { tikz }
7360         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7361         { \@@_error:n { tikz~key~without~tikz } } ,
7362     tikz .value_required:n = true ,
7363     fill .code:n =
7364         \tl_set_rescan:Nnn
7365         \l_@@_fill_tl
7366         { \char_set_catcode_other:N ! }
7367         { #1 } ,
7368     fill .value_required:n = true ,
7369     opacity .tl_set:N = \l_@@_opacity_tl ,
7370     opacity .value_required:n = true ,
7371     draw .code:n =
7372         \tl_set_rescan:Nnn
7373         \l_@@_draw_tl
7374         { \char_set_catcode_other:N ! }
7375         { #1 } ,
7376     draw .default:n = default ,
7377     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7378     rounded-corners .default:n = 4 pt ,

```

```

7379 color .code:n =
7380   \@@_color:n { #1 }
7381   \tl_set_rescan:Nnn
7382     \l_@@_draw_tl
7383     { \char_set_catcode_other:N ! }
7384     { #1 } ,
7385 borders .clist_set:N = \l_@@_borders_clist ,
7386 borders .value_required:n = true ,
7387 hvlines .meta:n = { vlines , hlines } ,
7388 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7389 vlines .default:n = true ,
7390 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7391 hlines .default:n = true ,
7392 line-width .dim_set:N = \l_@@_line_width_dim ,
7393 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7394 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7395 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7396 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7397 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7398   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7399 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7400   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7401 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7402   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7403 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7404 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7405 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7406 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7407 v-center .code:n = \str_set:Nn \l_@@_vpos_block_str { c } ,
7408 v-center .value_forbidden:n = true ,
7409 name .tl_set:N = \l_@@_block_name_str ,
7410 name .value_required:n = true ,
7411 name .initial:n = ,
7412 respect-arraystretch .code:n =
7413   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7414 respect-arraystretch .value_forbidden:n = true ,
7415 transparent .bool_set:N = \l_@@_transparent_bool ,
7416 transparent .default:n = true ,
7417 transparent .initial:n = false ,
7418 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7419 }

```

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.

```

7420 \cs_new_protected:Npn \@@_draw_blocks:
7421 {
7422   \cs_set_eq:NN \ialign \@@_old_ialign:
7423   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7424 }
7425 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7426 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7427   \int_zero_new:N \l_@@_last_row_int
7428   \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.

```

7429 \int_compare:nNnTF { #3 } > { 99 }
7430   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7431   { \int_set:Nn \l_@@_last_row_int { #3 } }
7432 \int_compare:nNnTF { #4 } > { 99 }
7433   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7434   { \int_set:Nn \l_@@_last_col_int { #4 } }
7435 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7436   {
7437     \bool_lazy_and:nnTF
7438       \l_@@_preamble_bool
7439       {
7440         \int_compare_p:n
7441           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7442       }
7443       {
7444         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7445         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7446         \@@_msg_redirect_name:nn { columns-not-used } { none }
7447       }
7448       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7449   }
7450   {
7451     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7452       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7453       { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7454   }
7455 }

```

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

```

7456 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7457   {

```

The group is for the keys.

```

7458   \group_begin:
7459   \int_compare:nNnT { #1 } = { #3 }
7460     { \str_set:Nn \l_@@_vpos_block_str { t } }
7461   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7462   \bool_if:NT \l_@@_vlines_block_bool
7463     {
7464       \tl_gput_right:Nx \g_nicematrix_code_after_tl
7465       {
7466         \@@_vlines_block:nnn
7467           { \exp_not:n { #5 } }
7468           { #1 - #2 }
7469           { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7470       }
7471     }
7472   \bool_if:NT \l_@@_hlines_block_bool
7473     {
7474       \tl_gput_right:Nx \g_nicematrix_code_after_tl
7475       {
7476         \@@_hlines_block:nnn
7477           { \exp_not:n { #5 } }
7478           { #1 - #2 }
7479           { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7480       }
7481     }
7482   \bool_if:NF \l_@@_transparent_bool

```

```

7483 {
7484   \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7485   {

```

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).

```

7486     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7487     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7488   }
7489 }

```

```

7490 \tl_if_empty:NF \l_@@_draw_tl
7491 {
7492   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7493   { \@@_error:n { hlines-with-color } }
7494 }

```

```

7495 \tl_if_empty:NF \l_@@_draw_tl
7496 {
7497   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7498   {
7499     \@@_stroke_block:nnn
7500     { \exp_not:n { #5 } } % #5 are the options
7501     { #1 - #2 }
7502     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7503   }
7504   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7505   { { #1 } { #2 } { #3 } { #4 } }
7506 }

```

```

7507 \clist_if_empty:NF \l_@@_borders_clist
7508 {
7509   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7510   {
7511     \@@_stroke_borders_block:nnn
7512     { \exp_not:n { #5 } }
7513     { #1 - #2 }
7514     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7515   }
7516 }

```

```

7517 \tl_if_empty:NF \l_@@_fill_tl
7518 {
7519   \tl_if_empty:NF \l_@@_opacity_tl
7520   {
7521     \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7522       {
7523         \tl_set:Nx \l_@@_fill_tl
7524         {
7525           [ opacity = \l_@@_opacity_tl ,
7526           \tl_tail:o \l_@@_fill_tl
7527         }
7528       }
7529       {
7530         \tl_set:Nx \l_@@_fill_tl
7531         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7532       }
7533     ]
7534     \tl_gput_right:Nx \g_@@_pre_code_before_tl
7535     {
7536       \exp_not:N \roundedrectanglecolor
7537       \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7538         { \l_@@_fill_tl }

```

```

7539         { { \l_@@_fill_tl } }
7540     { #1 - #2 }
7541     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7542     { \dim_use:N \l_@@_rounded_corners_dim }
7543 }
7544 }
7545 \seq_if_empty:NF \l_@@_tikz_seq
7546 {
7547     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7548     {
7549         \@@_block_tikz:nnnnn
7550         { #1 }
7551         { #2 }
7552         { \int_use:N \l_@@_last_row_int }
7553         { \int_use:N \l_@@_last_col_int }
7554         { \seq_use:Nn \l_@@_tikz_seq { , } }
7555     }
7556 }

7557 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7558 {
7559     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7560     {
7561         \@@_actually_diagbox:nnnnnn
7562         { #1 }
7563         { #2 }
7564         { \int_use:N \l_@@_last_row_int }
7565         { \int_use:N \l_@@_last_col_int }
7566         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7567     }
7568 }

7569 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7570 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & & four & five & \\\
six & & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

7571 \pgfpicture
7572 \pgfrememberpicturepositiononpagetrue
7573 \pgf@relevantforpicturesizefalse
7574 \@@_qpoint:n { row - #1 }
7575 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7576 \@@_qpoint:n { col - #2 }

```

```

7577 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7578 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7579 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7580 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7581 \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.

```

7582 \@@_pgf_rect_node:nnnnn
7583 { \@@_env: - #1 - #2 - block }
7584 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7585 \str_if_empty:NF \l_@@_block_name_str
7586 {
7587   \pgfnodealias
7588   { \@@_env: - \l_@@_block_name_str }
7589   { \@@_env: - #1 - #2 - block }
7590   \str_if_empty:NF \l_@@_name_str
7591   {
7592     \pgfnodealias
7593     { \l_@@_name_str - \l_@@_block_name_str }
7594     { \@@_env: - #1 - #2 - block }
7595   }
7596 }

```

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.

```

7597 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7598 {
7599   \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.

```

7600 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7601 {

```

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.

```

7602 \cs_if_exist:cT
7603 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7604 {
7605   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7606   {
7607     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7608     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7609   }
7610 }
7611 }

```

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.

```

7612 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7613 {
7614   \@@_qpoint:n { col - #2 }
7615   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7616 }
7617 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7618 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7619 {
7620   \cs_if_exist:cT
7621   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7622   {

```

```

7623         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7624         {
7625             \pgfpointanchor
7626             { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7627             { east }
7628             \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7629         }
7630     }
7631 }
7632 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7633 {
7634     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7635     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7636 }
7637 \@@_pgf_rect_node:nnnnn
7638 { \@@_env: - #1 - #2 - block - short }
7639 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7640 }

```

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.

```

7641 \bool_if:NT \l_@@_medium_nodes_bool
7642 {
7643     \@@_pgf_rect_node:nnn
7644     { \@@_env: - #1 - #2 - block - medium }
7645     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7646     {
7647         \pgfpointanchor
7648         { \@@_env:
7649             - \int_use:N \l_@@_last_row_int
7650             - \int_use:N \l_@@_last_col_int - medium
7651         }
7652         { south-east }
7653     }
7654 }

```

Now, we will put the label of the block.

```

7655 \bool_lazy_any:nTF
7656 {
7657     { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7658     { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7659     { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7660 }
7661 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7662     \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`.

```

7663     \bool_if:nT \g_@@_last_col_found_bool
7664     {
7665         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7666         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7667     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7668     \tl_set:Nx \l_tmpa_tl
7669     {
7670         \str_case:on \l_@@_vpos_block_str
7671         {
7672             c {
7673                 \str_case:on \l_@@_hpos_block_str
7674                 {

```

```

7675             c { center }
7676             l { west }
7677             r { east }
7678         }
7679
7680     }
7681     T {
7682         \str_case:on \l_@@_hpos_block_str
7683         {
7684             c { north }
7685             l { north~west }
7686             r { north~east }
7687         }
7688
7689     }
7690     B {
7691         \str_case:on \l_@@_hpos_block_str
7692         {
7693             c { south}
7694             l { south~west }
7695             r { south~east }
7696         }
7697
7698     }
7699 }
7700
7701 \pgftransformshift
7702 {
7703     \pgfpointanchor
7704     {
7705         \@@_env: - #1 - #2 - block
7706         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7707     }
7708     { \l_tmpa_tl }
7709 }
7710 \pgfset
7711 {
7712     inner~xsep = \c_zero_dim ,
7713     inner~ysep = \c_zero_dim
7714 }
7715 \pgfnode
7716 { rectangle }
7717 { \l_tmpa_tl }
7718 { \box_use_drop:N \l_@@_cell_box } { } { }
7719 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

7720 {
7721     \pgfextracty \l_tmpa_dim
7722     {
7723         \@@_qpoint:n
7724         {
7725             row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7726             - base
7727         }
7728     }
7729     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth } % added 2023-02-21

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

7730 \pgfpointanchor
7731 {
7732     \@@_env: - #1 - #2 - block
7733     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }

```

```

7734     }
7735     {
7736         \str_case:on \l_@@_hpos_block_str
7737         {
7738             c { center }
7739             l { west }
7740             r { east }
7741         }
7742     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7743     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7744     \pgfset { inner~sep = \c_zero_dim }
7745     \pgfnode
7746     { rectangle }
7747     {
7748         \str_case:on \l_@@_hpos_block_str
7749         {
7750             c { base }
7751             l { base~west }
7752             r { base~east }
7753         }
7754     }
7755     { \box_use_drop:N \l_@@_cell_box } { } { }
7756 }
7757 \endpgfpicture
7758 \group_end:
7759 }

```

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).

```

7760 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7761 {
7762     \group_begin:
7763     \tl_clear:N \l_@@_draw_tl
7764     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7765     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7766     \pgfpicture
7767     \pgfrememberpicturerepositiononpagetrue
7768     \pgf@relevantforpicturesizefalse
7769     \tl_if_empty:NF \l_@@_draw_tl
7770     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7771         \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7772         { \CT@arc@ }
7773         { \@@_color:o \l_@@_draw_tl }
7774     }
7775     \pgfsetcornersarced
7776     {
7777         \pgfpoint
7778         { \l_@@_rounded_corners_dim }
7779         { \l_@@_rounded_corners_dim }
7780     }
7781     \@@_cut_on_hyphen:w #2 \q_stop
7782     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7783     {
7784         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7785         {
7786             \@@_qpoint:n { row - \l_tmpa_tl }
7787             \dim_set_eq:NN \l_tmpb_dim \pgf@y

```

```

7788 \@@_qpoint:n { col - \l_tmpb_tl }
7789 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7790 \@@_cut_on_hyphen:w #3 \q_stop
7791 \int_compare:nNnT \l_tmpa_tl > \c@iRow
7792 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7793 \int_compare:nNnT \l_tmpb_tl > \c@jCol
7794 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7795 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7796 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7797 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7798 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7799 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7800 \pgfpathrectanglecorners
7801 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7802 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7803 \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7804 { \pgfusepathqstroke }
7805 { \pgfusepath { stroke } }
7806 }
7807 }
7808 \endpgfpicture
7809 \group_end:
7810 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7811 \keys_define:nn { NiceMatrix / BlockStroke }
7812 {
7813   color .tl_set:N = \l_@@_draw_tl ,
7814   draw .code:n =
7815     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7816   draw .default:n = default ,
7817   line-width .dim_set:N = \l_@@_line_width_dim ,
7818   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7819   rounded-corners .default:n = 4 pt
7820 }

```

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).

```

7821 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7822 {
7823   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7824   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7825   \@@_cut_on_hyphen:w #2 \q_stop
7826   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7827   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7828   \@@_cut_on_hyphen:w #3 \q_stop
7829   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7830   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7831   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7832   {
7833     \use:e
7834     {
7835       \@@_vline:n
7836       {
7837         position = ##1 ,
7838         start = \l_@@_tmpc_tl ,
7839         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7840         total-width = \dim_use:N \l_@@_line_width_dim
7841       }
7842     }
7843   }
7844 }

```

```

7845 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7846 {
7847   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7848   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7849   \@@_cut_on_hyphen:w #2 \q_stop
7850   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7851   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7852   \@@_cut_on_hyphen:w #3 \q_stop
7853   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7854   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7855   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7856   {
7857     \use:e
7858     {
7859       \@@_hline:n
7860       {
7861         position = ##1 ,
7862         start = \l_@@_tmpd_tl ,
7863         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7864         total-width = \dim_use:N \l_@@_line_width_dim
7865       }
7866     }
7867   }
7868 }

```

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).

```

7869 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7870 {
7871   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7872   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7873   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7874   { \@@_error:n { borders~forbidden } }
7875   {
7876     \tl_clear_new:N \l_@@_borders_tikz_tl
7877     \keys_set:nV
7878     { NiceMatrix / OnlyForTikzInBorders }
7879     \l_@@_borders_clist
7880     \@@_cut_on_hyphen:w #2 \q_stop
7881     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7882     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7883     \@@_cut_on_hyphen:w #3 \q_stop
7884     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7885     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7886     \@@_stroke_borders_block_i:
7887   }
7888 }
7889 \hook_gput_code:nnn { begindocument } { . }
7890 {
7891   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7892   {
7893     \c_@@_pgfortikzpicture_tl
7894     \@@_stroke_borders_block_ii:
7895     \c_@@_endpgfortikzpicture_tl
7896   }
7897 }
7898 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7899 {
7900   \pgfrememberpicturepositiononpagetrue
7901   \pgf@relevantforpicturesizefalse
7902   \CT@arc@

```

```

7903 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7904 \clist_if_in:NnT \l_@@_borders_clist { right }
7905   { \@@_stroke_vertical:n \l_tmpb_tl }
7906 \clist_if_in:NnT \l_@@_borders_clist { left }
7907   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7908 \clist_if_in:NnT \l_@@_borders_clist { bottom }
7909   { \@@_stroke_horizontal:n \l_tmpa_tl }
7910 \clist_if_in:NnT \l_@@_borders_clist { top }
7911   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7912 }
7913 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7914 {
7915   tikz .code:n =
7916     \cs_if_exist:NTF \tikzpicture
7917       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7918       { \@@_error:n { tikz~in~borders~without~tikz } } ,
7919   tikz .value_required:n = true ,
7920   top .code:n = ,
7921   bottom .code:n = ,
7922   left .code:n = ,
7923   right .code:n = ,
7924   unknown .code:n = \@@_error:n { bad~border }
7925 }

```

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).

```

7926 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7927 {
7928   \@@_qpoint:n \l_@@_tmpc_tl
7929   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7930   \@@_qpoint:n \l_tmpa_tl
7931   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7932   \@@_qpoint:n { #1 }
7933   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7934   {
7935     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7936     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7937     \pgfusepathqstroke
7938   }
7939   {
7940     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7941     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7942   }
7943 }

```

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).

```

7944 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7945 {
7946   \@@_qpoint:n \l_@@_tmpd_tl
7947   \clist_if_in:NnTF \l_@@_borders_clist { left }
7948     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7949     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7950   \@@_qpoint:n \l_tmpb_tl
7951   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7952   \@@_qpoint:n { #1 }
7953   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7954   {
7955     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7956     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7957     \pgfusepathqstroke
7958   }
7959   {

```

```

7960     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7961     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7962 }
7963 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7964 \keys_define:nn { NiceMatrix / BlockBorders }
7965 {
7966   borders .clist_set:N = \l_@@_borders_clist ,
7967   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7968   rounded-corners .default:n = 4 pt ,
7969   line-width .dim_set:N = \l_@@_line_width_dim
7970 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments `#1` and `#2` are the coordinates of the first cell and `#3` and `#4` the coordinates of the last cell of the block. `#5` is a comma-separated list of the Tikz keys used with the path. However, among those keys, you have added in `nicematrix` a special key `offset` (an offset for the rectangle of the block). That's why we have to extract that key first.

```

7971 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7972 {
7973   \begin { tikzpicture }
7974   \@@_clip_with_rounded_corners:
7975   \clist_map_inline:nn { #5 }
7976   {
7977     \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7978     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7979     (
7980       [
7981         xshift = \dim_use:N \l_@@_offset_dim ,
7982         yshift = - \dim_use:N \l_@@_offset_dim
7983       ]
7984       #1 -| #2
7985     )
7986     rectangle
7987     (
7988       [
7989         xshift = - \dim_use:N \l_@@_offset_dim ,
7990         yshift = \dim_use:N \l_@@_offset_dim
7991       ]
7992       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7993     ) ;
7994   }
7995   \end { tikzpicture }
7996 }
7997 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7998 \keys_define:nn { NiceMatrix / SpecialOffset }
7999 { offset .dim_set:N = \l_@@_offset_dim }

```

## 28 How to draw the dotted lines transparently

```

8000 \cs_set_protected:Npn \@@_renew_matrix:
8001 {
8002   \RenewDocumentEnvironment { pmatrix } { }
8003   { \pNiceMatrix }
8004   { \endpNiceMatrix }
8005   \RenewDocumentEnvironment { vmatrix } { }
8006   { \vNiceMatrix }

```

```

8007     { \endvNiceMatrix }
8008 \RenewDocumentEnvironment { Vmatrix } { }
8009     { \VNiceMatrix }
8010     { \endVNiceMatrix }
8011 \RenewDocumentEnvironment { bmatrix } { }
8012     { \bNiceMatrix }
8013     { \endbNiceMatrix }
8014 \RenewDocumentEnvironment { Bmatrix } { }
8015     { \BNiceMatrix }
8016     { \endBNiceMatrix }
8017 }

```

## 29 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

8018 \keys_define:nn { NiceMatrix / Auto }
8019 {
8020     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8021     columns-type .value_required:n = true ,
8022     l .meta:n = { columns-type = l } ,
8023     r .meta:n = { columns-type = r } ,
8024     c .meta:n = { columns-type = c } ,
8025     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8026     delimiters / color .value_required:n = true ,
8027     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8028     delimiters / max-width .default:n = true ,
8029     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
8030     delimiters .value_required:n = true ,
8031     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8032     rounded-corners .default:n = 4 pt
8033 }
8034 \NewDocumentCommand \AutoNiceMatrixWithDelims
8035 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8036 { \l_@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8037 \cs_new_protected:Npn \l_@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8038 {

```

The group is for the protection of the keys.

```

8039 \group_begin:
8040 \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
8041 \use:e
8042 {
8043     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8044     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8045     [ \exp_not:o \l_tmpa_tl ]
8046 }
8047 \int_if_zero:nT \l_@@_first_row_int
8048 {
8049     \int_if_zero:nT \l_@@_first_col_int { & }
8050     \prg_replicate:nn { #4 - 1 } { & }
8051     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8052 }
8053 \prg_replicate:nn { #3 }
8054 {
8055     \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).

```

8056     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8057     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\

```

```

8058     }
8059     \int_compare:nNtT \l_@@_last_row_int > { -2 }
8060     {
8061         \int_if_zero:nT \l_@@_first_col_int { & }
8062         \prg_replicate:nn { #4 - 1 } { & }
8063         \int_compare:nNtT \l_@@_last_col_int > { -1 } { & } \\
8064     }
8065     \end { NiceArrayWithDelims }
8066     \group_end:
8067 }

8068 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8069 {
8070     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8071     {
8072         \bool_gset_true:N \g_@@_delims_bool
8073         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8074         \AutoNiceMatrixWithDelims { #2 } { #3 }
8075     }
8076 }

8077 \@@_define_com:nnn p ( )
8078 \@@_define_com:nnn b [ ]
8079 \@@_define_com:nnn v | |
8080 \@@_define_com:nnn V \| \|
8081 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8082 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8083 {
8084     \group_begin:
8085     \bool_gset_false:N \g_@@_delims_bool
8086     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8087     \group_end:
8088 }

```

## 30 The redefinition of the command `\dotfill`

```

8089 \cs_set_eq:NN \@@_old_dotfill \dotfill
8090 \cs_new_protected:Npn \@@_dotfill:
8091 {

```

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}`).

```

8092     \@@_old_dotfill
8093     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8094 }

```

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`.

```

8095 \cs_new_protected:Npn \@@_dotfill_i:
8096 { \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

## 31 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.

```

8097 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8098 {
8099   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8100   {
8101     \@@_actually_diagbox:nnnnnn
8102     { \int_use:N \c@iRow }
8103     { \int_use:N \c@jCol }
8104     { \int_use:N \c@iRow }
8105     { \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.

```

8106     { \g_@@_row_style_tl \exp_not:n { #1 } }
8107     { \g_@@_row_style_tl \exp_not:n { #2 } }
8108   }

```

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`.

```

8109   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8110   {
8111     { \int_use:N \c@iRow }
8112     { \int_use:N \c@jCol }
8113     { \int_use:N \c@iRow }
8114     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8115     { }
8116   }
8117 }

```

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.

```

8118 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8119 {
8120   \pgfpicture
8121   \pgf@relevantforpicturesizefalse
8122   \pgfrememberpicturepositiononpagetrue
8123   \@@_qpoint:n { row - #1 }
8124   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8125   \@@_qpoint:n { col - #2 }
8126   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8127   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8128   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8129   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8130   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8131   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8132   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8133   {

```

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.

```

8134   \CT@arc@
8135   \pgfsetroundcap
8136   \pgfusepathqstroke
8137 }
8138 \pgfset { inner~sep = 1 pt }
8139 \pgfscope
8140 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }

```

```

8141 \pgfnode { rectangle } { south-west }
8142 {
8143   \begin { minipage } { 20 cm }
8144   \@@_math_toggle: #5 \@@_math_toggle:
8145   \end { minipage }
8146 }
8147 { }
8148 { }
8149 \endpgfscope
8150 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8151 \pgfnode { rectangle } { north-east }
8152 {
8153   \begin { minipage } { 20 cm }
8154   \raggedleft
8155   \@@_math_toggle: #6 \@@_math_toggle:
8156   \end { minipage }
8157 }
8158 { }
8159 { }
8160 \endpgfpicture
8161 }

```

## 32 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. 83.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8162 \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 `\`.

```

8163 \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`.

```

8164 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8165 {
8166   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8167   \@@_CodeAfter_iv:n
8168 }

```

We catch the argument of the command `\end` (in `#1`).

```

8169 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8170 {

```

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.

```

8171   \str_if_eq:eeTF \@currenvir { #1 }
8172   { \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`.

```

8173   {
8174     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8175     \@@_CodeAfter_ii:n
8176   }
8177 }

```

### 33 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.

```
8178 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8179 {
8180   \pgfpicture
8181   \pgfrememberpicturepositiononpagetrue
8182   \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.

```
8183   \@@_qpoint:n { row - 1 }
8184   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8185   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8186   \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).

```
8187   \bool_if:nTF { #3 }
8188   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8189   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8190   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8191   {
8192     \cs_if_exist:cT
8193     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8194     {
8195       \pgfpointanchor
8196       { \@@_env: - ##1 - #2 }
8197       { \bool_if:nTF { #3 } { west } { east } }
8198       \dim_set:Nn \l_tmpa_dim
8199       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8200     }
8201   }
```

Now we can put the delimiter with a node of PGF.

```
8202   \pgfset { inner~sep = \c_zero_dim }
8203   \dim_zero:N \nulldelimiterspace
8204   \pgftransformshift
8205   {
8206     \pgfpoint
8207     { \l_tmpa_dim }
8208     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8209   }
8210   \pgfnode
8211   { rectangle }
8212   { \bool_if:nTF { #3 } { east } { west } }
8213   {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```
8214     \nullfont
8215     \c_math_toggle_token
8216     \@@_color:o \l_@@_delimiters_color_tl
8217     \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```

8218     \vcenter
8219     {
8220         \nullfont
8221         \hrule \@height
8222             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8223             \@depth \c_zero_dim
8224             \@width \c_zero_dim
8225     }
8226     \bool_if:nTF { #3 } { \right . } { \right #1 }
8227     \c_math_toggle_token
8228 }
8229 { }
8230 { }
8231 \endpgfpicture
8232 }

```

## 34 The command \SubMatrix

```

8233 \keys_define:nn { NiceMatrix / sub-matrix }
8234 {
8235     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8236     extra-height .value_required:n = true ,
8237     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8238     left-xshift .value_required:n = true ,
8239     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8240     right-xshift .value_required:n = true ,
8241     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8242     xshift .value_required:n = true ,
8243     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8244     delimiters / color .value_required:n = true ,
8245     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8246     slim .default:n = true ,
8247     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8248     hlines .default:n = all ,
8249     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8250     vlines .default:n = all ,
8251     hvlines .meta:n = { hlines, vlines } ,
8252     hvlines .value_forbidden:n = true
8253 }
8254 \keys_define:nn { NiceMatrix }
8255 {
8256     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8257     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8258     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8259     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8260 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8261 \keys_define:nn { NiceMatrix / SubMatrix }
8262 {
8263     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8264     delimiters / color .value_required:n = true ,
8265     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8266     hlines .default:n = all ,
8267     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8268     vlines .default:n = all ,
8269     hvlines .meta:n = { hlines, vlines } ,
8270     hvlines .value_forbidden:n = true ,
8271     name .code:n =

```

```

8272 \tl_if_empty:nTF { #1 }
8273 { \@@_error:n { Invalid-name } }
8274 {
8275   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8276   {
8277     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8278     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8279     {
8280       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8281       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8282     }
8283   }
8284   { \@@_error:n { Invalid-name } }
8285 } ,
8286 name .value_required:n = true ,
8287 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8288 rules .value_required:n = true ,
8289 code .tl_set:N = \l_@@_code_tl ,
8290 code .value_required:n = true ,
8291 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8292 }

8293 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8294 {
8295   \peek_remove_spaces:n
8296   {
8297     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8298     {
8299       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8300       [
8301         delimiters / color = \l_@@_delimiters_color_tl ,
8302         hlines = \l_@@_submatrix_hlines_clist ,
8303         vlines = \l_@@_submatrix_vlines_clist ,
8304         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8305         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8306         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8307         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8308         #5
8309       ]
8310     }
8311     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8312   }
8313 }

8314 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8315 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8316 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8317 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8318 {
8319   \seq_gput_right:Nx \g_@@_submatrix_seq
8320   {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8321 { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8322 { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8323 { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8324 { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8325 }
8326 }

```

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`.

```

8327 \hook_gput_code:nnn { begindocument } { . }
8328 {
8329   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8330   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8331   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8332     {
8333       \peek_remove_spaces:n
8334       {
8335         \@@_sub_matrix:nnnnnnn
8336         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8337       }
8338     }
8339 }

```

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).

```

8340 \NewDocumentCommand \@@_compute_i_j:nn
8341 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8342 { \@@_compute_i_j:nnnn #1 #2 }
8343 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8344 {
8345   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8346   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8347   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8348   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8349   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8350     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8351   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8352     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8353   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8354     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8355   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8356     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8357 }
8358 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8359 {
8360   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8361 \@@_compute_i_j:nn { #2 } { #3 }
8362 \int_compare:NnNT \l_@@_first_i_tl = \l_@@_last_i_tl
8363   { \cs_set_nopar:Npn \arraystretch { 1 } }
8364 \bool_lazy_or:nnTF
8365   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8366   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8367   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8368   {
8369     \str_clear_new:N \l_@@_submatrix_name_str
8370     \keys_set:nn { NiceMatrix / SubMatrix } { #5 }

```

```

8371 \pgfpicture
8372 \pgfrememberpicturepositiononpagetrue
8373 \pgf@relevantforpicturesizefalse
8374 \pgfset { inner~sep = \c_zero_dim }
8375 \dim_set_eq:Nn \l_@@_x_initial_dim \c_max_dim
8376 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currification.

8377 \bool_if:NTF \l_@@_submatrix_slim_bool
8378 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8379 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8380 {
8381   \cs_if_exist:cT
8382   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8383   {
8384     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8385     \dim_set:Nn \l_@@_x_initial_dim
8386     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8387   }
8388   \cs_if_exist:cT
8389   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8390   {
8391     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8392     \dim_set:Nn \l_@@_x_final_dim
8393     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8394   }
8395 }
8396 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8397 { \@@_error:nn { Impossible~delimiter } { left } }
8398 {
8399   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8400   { \@@_error:nn { Impossible~delimiter } { right } }
8401   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8402 }
8403 \endpgfpicture
8404 }
8405 \group_end:
8406 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8407 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8408 {
8409   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8410   \dim_set:Nn \l_@@_y_initial_dim
8411   {
8412     \fp_to_dim:n
8413     {
8414       \pgf@y
8415       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8416     }
8417     } % modified 6.13c
8418   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8419   \dim_set:Nn \l_@@_y_final_dim
8420   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8421   } % modified 6.13c
8422   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8423   {
8424     \cs_if_exist:cT
8425     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8426     {
8427       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8428       \dim_set:Nn \l_@@_y_initial_dim
8429       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }

```

```

8430     }
8431     \cs_if_exist:cT
8432     { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8433     {
8434         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8435         \dim_set:Nn \l_@@_y_final_dim
8436         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8437     }
8438 }
8439 \dim_set:Nn \l_tmpa_dim
8440 {
8441     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8442     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8443 }
8444 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8445     \group_begin:
8446     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8447     \@@_set_CT@arc@:o \l_@@_rules_color_tl
8448     \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`.

```

8449     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8450     {
8451         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8452         {
8453             \int_compare:nNnT
8454             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8455             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8456                 \@@_qpoint:n { col - ##1 }
8457                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8458                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8459                 \pgfusepathqstroke
8460             }
8461         }
8462     }

```

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.

```

8463     \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8464     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8465     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8466     {
8467         \bool_lazy_and:nnTF
8468         { \int_compare_p:nNn { ##1 } > \c_zero_int }
8469         {
8470             \int_compare_p:nNn
8471             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8472         {
8473             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8474             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8475             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8476             \pgfusepathqstroke
8477         }
8478         { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8479     }

```

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.

```

8480 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8481 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8482 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8483 {
8484   \bool_lazy_and:nnTF
8485   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8486   {
8487     \int_compare_p:nNn
8488     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8489   {
8490     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8491   \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
8492   \dim_set:Nn \l_tmpa_dim
8493   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8494   \str_case:nn { #1 }
8495   {
8496     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8497     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8498     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8499     }
8500   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

8501   \dim_set:Nn \l_tmpb_dim
8502   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8503   \str_case:nn { #2 }
8504   {
8505     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8506     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8507     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8508   }
8509   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8510   \pgfusepathqstroke
8511   \group_end:
8512 }
8513 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8514 }

```

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).

```

8515 \str_if_empty:NF \l_@@_submatrix_name_str
8516 {
8517   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8518   \l_@@_x_initial_dim \l_@@_y_initial_dim
8519   \l_@@_x_final_dim \l_@@_y_final_dim
8520 }
8521 \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`.

```

8522 \begin { pgfscope }
8523 \pgftransformshift
8524 {
8525   \pgfpoint
8526   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8527   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8528 }

```

```

8529 \str_if_empty:NTF \l_@@_submatrix_name_str
8530 { \@@_node_left:nn #1 { } }
8531 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8532 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8533 \pgftransformshift
8534 {
8535   \pgfpoint
8536   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8537   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8538 }
8539 \str_if_empty:NTF \l_@@_submatrix_name_str
8540 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8541 {
8542   \@@_node_right:nnnn #2
8543   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8544 }
8545 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8546 \flag_clear_new:n { nicematrix }
8547 \l_@@_code_tl
8548 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the  $i$  and  $j$  in specifications of nodes of the forms  $i-j$ ,  $\text{row-}i$ ,  $\text{col-}j$  and  $i-|j$  refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8549 \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.

```

8550 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8551 {
8552   \use:e
8553   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8554 }

```

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 -.

```

8555 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8556 { #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.

```

8557 \tl_const:Nn \c_@@_integers_alist_tl
8558 {
8559   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8560   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8561   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8562   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8563 }
8564 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8565 {

```

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`.

```

8566 \tl_if_empty:nTF { #2 }
8567 {
8568   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8569   {
8570     \flag_raise:n { nicematrix }
8571     \int_if_even:nTF { \flag_height:n { nicematrix } }
8572     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8573     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8574   }
8575   { #1 }
8576 }

```

If there is an hyphen, we have to see whether we have a node of the form  $i-j$ , row- $i$  or col- $j$ .

```

8577 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8578 }

```

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`).

```

8579 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8580 {
8581   \str_case:nnF { #1 }
8582   {
8583     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8584     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8585   }

```

Now the case of a node of the form  $i-j$ .

```

8586 {
8587   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8588   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8589 }
8590 }

```

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`).

```

8591 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8592 {
8593   \pgfnode
8594   { rectangle }
8595   { east }
8596   {
8597     \nullfont
8598     \c_math_toggle_token
8599     \@@_color:o \l_@@_delimiters_color_tl
8600     \left #1
8601     \vcenter
8602     {
8603       \nullfont
8604       \hrule \@height \l_tmpa_dim
8605       \@depth \c_zero_dim
8606       \@width \c_zero_dim
8607     }
8608     \right .
8609     \c_math_toggle_token
8610   }
8611   { #2 }

```

```

8612 { }
8613 }

```

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.

```

8614 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8615 {
8616   \pgfnode
8617   { rectangle }
8618   { west }
8619   {
8620     \nullfont
8621     \c_math_toggle_token
8622     \@@_color:o \l_@@_delimiters_color_tl
8623     \left .
8624     \vcenter
8625     {
8626       \nullfont
8627       \hrule \@height \l_tmpa_dim
8628       \@depth \c_zero_dim
8629       \@width \c_zero_dim
8630     }
8631     \right #1
8632     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8633     ^ { \smash { #4 } }
8634     \c_math_toggle_token
8635   }
8636   { #2 }
8637   { }
8638 }

```

## 35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8639 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8640 {
8641   \peek_remove_spaces:n
8642   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8643 }
8644 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8645 {
8646   \peek_remove_spaces:n
8647   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8648 }
8649 \keys_define:nn { NiceMatrix / Brace }
8650 {
8651   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8652   left-shorten .default:n = true ,
8653   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8654   shorten .meta:n = { left-shorten , right-shorten } ,
8655   right-shorten .default:n = true ,
8656   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8657   yshift .value_required:n = true ,
8658   yshift .initial:n = \c_zero_dim ,
8659   color .tl_set:N = \l_tmpa_tl ,
8660   color .value_required:n = true ,

```

```

8661     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8662 }

```

#1 is the first cell of the rectangle (with the syntax  $i-j$ ; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to `under` or `over`.

```

8663 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8664 {
8665     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8666     \@@_compute_i_j:nn { #1 } { #2 }
8667     \bool_lazy_or:nnTF
8668     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8669     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8670     {
8671         \str_if_eq:nnTF { #5 } { under }
8672         { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8673         { \@@_error:nn { Construct~too~large } { \OverBrace } }
8674     }
8675     {
8676         \tl_clear:N \l_tmpa_tl
8677         \keys_set:nn { NiceMatrix / Brace } { #4 }
8678         \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8679         \pgfpicture
8680         \pgfrememberpicturepositiononpagetrue
8681         \pgf@relevantforpicturesizefalse
8682         \bool_if:NT \l_@@_brace_left_shorten_bool
8683         {
8684             \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8685             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8686             {
8687                 \cs_if_exist:cT
8688                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8689                 {
8690                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8691                     \dim_set:Nn \l_@@_x_initial_dim
8692                     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8693                 }
8694             }
8695         }
8696         \bool_lazy_or:nnT
8697         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8698         { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8699         {
8700             \@@_qpoint:n { col - \l_@@_first_j_tl }
8701             \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8702         }
8703         \bool_if:NT \l_@@_brace_right_shorten_bool
8704         {
8705             \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8706             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8707             {
8708                 \cs_if_exist:cT
8709                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8710                 {
8711                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8712                     \dim_set:Nn \l_@@_x_final_dim
8713                     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8714                 }
8715             }
8716         }
8717         \bool_lazy_or:nnT
8718         { \bool_not_p:n \l_@@_brace_right_shorten_bool }

```

```

8719         { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8720     {
8721         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8722         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8723     }
8724     \pgfset { inner~sep = \c_zero_dim }
8725     \str_if_eq:nnTF { #5 } { under }
8726     { \@@_underbrace_i:n { #3 } }
8727     { \@@_overbrace_i:n { #3 } }
8728     \endpgfpicture
8729 }
8730 \group_end:
8731 }

```

The argument is the text to put above the brace.

```

8732 \cs_new_protected:Npn \@@_overbrace_i:n #1
8733 {
8734     \@@_qpoint:n { row - \l_@@_first_i_tl }
8735     \pgftransformshift
8736     {
8737         \pgfpoint
8738         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8739         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8740     }
8741     \pgfnode
8742     { rectangle }
8743     { south }
8744     {
8745         \vtop
8746         {
8747             \group_begin:
8748             \everycr { }
8749             \halign
8750             {
8751                 \hfil ## \hfil \crcr
8752                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8753                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8754                 \c_math_toggle_token
8755                 \overbrace
8756                 {
8757                     \hbox_to_wd:nn
8758                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8759                     { }
8760                 }
8761                 \c_math_toggle_token
8762                 \cr
8763             }
8764             \group_end:
8765         }
8766     }
8767     { }
8768     { }
8769 }

```

The argument is the text to put under the brace.

```

8770 \cs_new_protected:Npn \@@_underbrace_i:n #1
8771 {
8772     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8773     \pgftransformshift
8774     {
8775         \pgfpoint
8776         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8777         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }

```

```

8778     }
8779 \pgfnode
8780 { rectangle }
8781 { north }
8782 {
8783     \group_begin:
8784     \everycr { }
8785     \vbox
8786     {
8787         \halign
8788         {
8789             \hfil ## \hfil \crcr
8790             \c_math_toggle_token
8791             \underbrace
8792             {
8793                 \hbox_to_wd:nn
8794                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8795                 { }
8796             }
8797             \c_math_toggle_token
8798             \cr
8799             \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8800             \@@_math_toggle: #1 \@@_math_toggle: \cr
8801         }
8802     }
8803     \group_end:
8804 }
8805 { }
8806 { }
8807 }

```

## 36 The command TikzEveryCell

```

8808 \bool_new:N \l_@@_not_empty_bool
8809 \bool_new:N \l_@@_empty_bool
8810
8811 \keys_define:nn { NiceMatrix / TikzEveryCell }
8812 {
8813     not-empty .code:n =
8814         \bool_lazy_or:nnTF
8815         \l_@@_in_code_after_bool
8816         \g_@@_recreate_cell_nodes_bool
8817         { \bool_set_true:N \l_@@_not_empty_bool }
8818         { \@@_error:n { detection~of~empty~cells } } ,
8819     not-empty .value_forbidden:n = true ,
8820     empty .code:n =
8821         \bool_lazy_or:nnTF
8822         \l_@@_in_code_after_bool
8823         \g_@@_recreate_cell_nodes_bool
8824         { \bool_set_true:N \l_@@_empty_bool }
8825         { \@@_error:n { detection~of~empty~cells } } ,
8826     empty .value_forbidden:n = true ,
8827     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8828 }
8829
8830
8831 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8832 {
8833     \IfPackageLoadedTF { tikz }

```

```

8834 {
8835     \group_begin:
8836     \keys_set:nn { NiceMatrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

8837     \tl_set:Nn \l_tmpa_tl { { #2 } }
8838     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8839     { \@@_for_a_block:nnnnn ##1 }
8840     \@@_all_the_cells:
8841     \group_end:
8842 }
8843 { \@@_error:n { TikzEveryCell~without~tikz } }
8844 }
8845
8846 \tl_new:N \@@_i_tl
8847 \tl_new:N \@@_j_tl
8848
8849 \cs_new_protected:Nn \@@_all_the_cells:
8850 {
8851     \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8852     {
8853         \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8854         {
8855             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8856             {
8857                 \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8858                 { \@@_i_tl - \@@_j_tl }
8859                 {
8860                     \bool_set_false:N \l_tmpa_bool
8861                     \cs_if_exist:cTF
8862                     { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8863                     {
8864                         \bool_if:NF \l_@@_empty_bool
8865                         { \bool_set_true:N \l_tmpa_bool }
8866                     }
8867                     {
8868                         \bool_if:NF \l_@@_not_empty_bool
8869                         { \bool_set_true:N \l_tmpa_bool }
8870                     }
8871                     \bool_if:NT \l_tmpa_bool
8872                     {
8873                         \@@_block_tikz:nnnnV
8874                         \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8875                     }
8876                 }
8877             }
8878         }
8879     }
8880 }
8881
8882 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8883 {
8884     \bool_if:NF \l_@@_empty_bool
8885     {
8886         \@@_block_tikz:nnnnV
8887         { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8888     }
8889     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8890 }
8891
8892 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8893 {
8894     \int_step_inline:nnn { #1 } { #3 }

```

```

8895 {
8896   \int_step_inline:nnn { #2 } { #4 }
8897   { \cs_set:cpn { cell - ##1 - #####1 } { } }
8898 }
8899 }

```

## 37 The command \ShowCellNames

```

8900 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8901 {
8902   \dim_zero_new:N \g_@@_tmpc_dim
8903   \dim_zero_new:N \g_@@_tmpd_dim
8904   \dim_zero_new:N \g_@@_tmpe_dim
8905   \int_step_inline:nn \c@iRow
8906   {
8907     \begin { pgfpicture }
8908     \@@_qpoint:n { row - ##1 }
8909     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8910     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8911     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8912     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8913     \bool_if:NTF \l_@@_in_code_after_bool
8914     \end { pgfpicture }
8915     \int_step_inline:nn \c@jCol
8916     {
8917       \hbox_set:Nn \l_tmpa_box
8918       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8919       \begin { pgfpicture }
8920       \@@_qpoint:n { col - #####1 }
8921       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8922       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8923       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8924       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8925       \endpgfpicture
8926       \end { pgfpicture }
8927       \fp_set:Nn \l_tmpa_fp
8928       {
8929         \fp_min:nn
8930         {
8931           \fp_min:nn
8932           {
8933             \dim_ratio:nn
8934             { \g_@@_tmpd_dim }
8935             { \box_wd:N \l_tmpa_box }
8936           }
8937           {
8938             \dim_ratio:nn
8939             { \g_tmpb_dim }
8940             { \box_ht_plus_dp:N \l_tmpa_box }
8941           }
8942         }
8943         { 1.0 }
8944       }
8945       \box_scale:Nnn \l_tmpa_box
8946       { \fp_use:N \l_tmpa_fp }
8947       { \fp_use:N \l_tmpa_fp }
8948       \pgfpicture
8949       \pgfrememberpicturerepositiononpagetrue
8950       \pgf@relevantforpicturesizefalse
8951       \pgftransformshift
8952       {
8953         \pgfpoint

```

```

8954         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8955         { \dim_use:N \g_tmpa_dim }
8956     }
8957     \pgfnode
8958     { rectangle }
8959     { center }
8960     { \box_use:N \l_tmpa_box }
8961     { }
8962     { }
8963     \endpgfpicture
8964 }
8965 }
8966 }
8967 \NewDocumentCommand \@@_ShowCellNames { }
8968 {
8969     \bool_if:NT \l_@@_in_code_after_bool
8970     {
8971         \pgfpicture
8972         \pgfrememberpicturepositiononpagetrue
8973         \pgf@relevantforpicturesizefalse
8974         \pgfpathrectanglecorners
8975         { \@@_qpoint:n { 1 } }
8976         {
8977             \@@_qpoint:n
8978             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8979         }
8980         \pgfsetfillopacity { 0.75 }
8981         \pgfsetfillcolor { white }
8982         \pgfusepathqfill
8983         \endpgfpicture
8984     }
8985     \dim_zero_new:N \g_@@_tmpc_dim
8986     \dim_zero_new:N \g_@@_tmpd_dim
8987     \dim_zero_new:N \g_@@_tmpe_dim
8988     \int_step_inline:nn \c@iRow
8989     {
8990         \bool_if:NTF \l_@@_in_code_after_bool
8991         {
8992             \pgfpicture
8993             \pgfrememberpicturepositiononpagetrue
8994             \pgf@relevantforpicturesizefalse
8995         }
8996         { \begin { pgfpicture } }
8997         \@@_qpoint:n { row - ##1 }
8998         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8999         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9000         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9001         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9002         \bool_if:NTF \l_@@_in_code_after_bool
9003         { \endpgfpicture }
9004         { \end { pgfpicture } }
9005         \int_step_inline:nn \c@jCol
9006         {
9007             \hbox_set:Nn \l_tmpa_box
9008             {
9009                 \normalfont \Large \sffamily \bfseries
9010                 \bool_if:NTF \l_@@_in_code_after_bool
9011                 { \color { red } }
9012                 { \color { red ! 50 } }
9013                 ##1 - ###1
9014             }
9015             \bool_if:NTF \l_@@_in_code_after_bool
9016             {

```

```

9017         \pgfpicture
9018         \pgfrememberpicturerepositiononpagetrue
9019         \pgf@relevantforpicturesizefalse
9020     }
9021     { \begin { pgfpicture } }
9022     @@_qpoint:n { col - ####1 }
9023     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9024     @@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9025     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9026     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9027     \bool_if:NTF \l_@@_in_code_after_bool
9028     { \endpgfpicture }
9029     { \end { pgfpicture } }
9030     \fp_set:Nn \l_tmpa_fp
9031     {
9032         \fp_min:nn
9033         {
9034             \fp_min:nn
9035             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9036             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9037         }
9038         { 1.0 }
9039     }
9040     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9041     \pgfpicture
9042     \pgfrememberpicturerepositiononpagetrue
9043     \pgf@relevantforpicturesizefalse
9044     \pgftransformshift
9045     {
9046         \pgfpoint
9047         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9048         { \dim_use:N \g_tmpa_dim }
9049     }
9050     \pgfnode
9051     { rectangle }
9052     { center }
9053     { \box_use:N \l_tmpa_box }
9054     { }
9055     { }
9056     \endpgfpicture
9057 }
9058 }
9059 }

```

## 38 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```
9060 \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.

```

9061 \bool_new:N \g_@@_footnote_bool
9062 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9063 {
9064     The~key~'\l_keys_key_str'~is~unknown. \

```

```

9065     That~key~will~be~ignored. \\
9066     For~a~list~of~the~available~keys,~type~H~<return>.
9067   }
9068   {
9069     The~available~keys~are~(in~alphabetic~order):~
9070     footnote,~
9071     footnotehyper,~
9072     messages~for~Overleaf,~
9073     no~test~for~array,~
9074     renew~dots,~and~
9075     renew~matrix.
9076   }
9077 \keys_define:nn { NiceMatrix / Package }
9078   {
9079     renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9080     renew~dots .value_forbidden:n = true ,
9081     renew~matrix .code:n = \@@_renew_matrix: ,
9082     renew~matrix .value_forbidden:n = true ,
9083     messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9084     footnote .bool_set:N = \g_@@_footnote_bool ,
9085     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9086     no~test~for~array .bool_set:N = \g_@@_no_test_for_array_bool ,
9087     no~test~for~array .default:n = true ,
9088     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9089   }
9090 \ProcessKeysOptions { NiceMatrix / Package }

9091 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9092   {
9093     You~can't~use~the~option~'footnote'~because~the~package~
9094     footnotehyper~has~already~been~loaded.~
9095     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9096     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9097     of~the~package~footnotehyper.\\
9098     The~package~footnote~won't~be~loaded.
9099   }

9100 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9101   {
9102     You~can't~use~the~option~'footnotehyper'~because~the~package~
9103     footnote~has~already~been~loaded.~
9104     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9105     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9106     of~the~package~footnote.\\
9107     The~package~footnotehyper~won't~be~loaded.
9108   }

9109 \bool_if:NT \g_@@_footnote_bool
9110   {
The class beamer has its own system to extract footnotes and that's why we have nothing to do if
beamer is used.
9111     \IfClassLoadedTF { beamer }
9112       { \bool_set_false:N \g_@@_footnote_bool }
9113       {
9114         \IfPackageLoadedTF { footnotehyper }
9115           { \@@_error:n { footnote~with~footnotehyper~package } }
9116           { \usepackage { footnote } }
9117       }
9118     }

9119 \bool_if:NT \g_@@_footnotehyper_bool
9120   {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9121 \IfClassLoadedTF { beamer }
9122 { \bool_set_false:N \g_@@_footnote_bool }
9123 {
9124   \IfPackageLoadedTF { footnote }
9125   { \@@_error:n { footnotehyper~with~footnote~package } }
9126   { \usepackage { footnotehyper } }
9127 }
9128 \bool_set_true:N \g_@@_footnote_bool
9129 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

## 39 About the package underscore

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```

9130 \bool_new:N \l_@@_underscore_loaded_bool
9131 \IfPackageLoadedTF { underscore }
9132 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9133 { }
9134 \hook_gput_code:nnn { begindocument } { . }
9135 {
9136   \bool_if:NF \l_@@_underscore_loaded_bool
9137   {
9138     \IfPackageLoadedTF { underscore }
9139     { \@@_error:n { underscore~after~nicematrix } }
9140     { }
9141   }
9142 }

```

## 40 Error messages of the package

```

9143 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9144 { \str_const:Nn \c_@@_available_keys_str { } }
9145 {
9146   \str_const:Nn \c_@@_available_keys_str
9147   { For~a~list~of~the~available~keys,~type~H~<return>. }
9148 }
9149 \seq_new:N \g_@@_types_of_matrix_seq
9150 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9151 {
9152   NiceMatrix ,
9153   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9154 }
9155 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9156 { \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:NoTF` 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`.

```

9157 \cs_new_protected:Npn \@@_error_too_much_cols:
9158 {
9159   \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9160   {
9161     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9162     { \@@_fatal:n { too-much-cols-for-matrix } }
9163     {
9164       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9165       { \@@_fatal:n { too-much-cols-for-matrix } }
9166       {
9167         \bool_if:NF \l_@@_last_col_without_value_bool
9168         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9169       }
9170     }
9171   }
9172   { \@@_fatal:nn { too-much-cols-for-array } }
9173 }

```

The following command must *not* be protected since it's used in an error message.

```

9174 \cs_new:Npn \@@_message_hdotsfor:
9175 {
9176   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9177   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
9178 }
9179 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9180 {
9181   Incompatible-options.\\
9182   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9183   The-output-will-not-be-reliable.
9184 }
9185 \@@_msg_new:nn { negative-weight }
9186 {
9187   Negative-weight.\\
9188   The-weight-of~the~'X'~columns~must~be~positive~and~you~have~used~
9189   the~value~'\int_use:N \l_@@_weight_int'.\\
9190   The~absolute~value~will~be~used.
9191 }
9192 \@@_msg_new:nn { last-col-not-used }
9193 {
9194   Column~not~used.\\
9195   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9196   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9197 }
9198 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9199 {
9200   Too-much-columns.\\
9201   In~the~row~\int_eval:n { \c@iRow },~
9202   you~try~to~use~more~columns~
9203   than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\\
9204   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9205   (plus~the~exterior~columns).~This~error~is~fatal.
9206 }
9207 \@@_msg_new:nn { too-much-cols-for-matrix }
9208 {
9209   Too-much-columns.\\
9210   In~the~row~\int_eval:n { \c@iRow },~
9211   you~try~to~use~more~columns~than~allowed~by~your~
9212   \@@_full_name_env:.~\@@_message_hdotsfor:\\ Recall~that~the~maximal~
9213   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9214   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~

```

```

9215     Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9216     \token_to_str:N \setcounter\ to~change~that~value).~
9217     This-error-is-fatal.
9218 }

9219 \@@_msg_new:nn { too-much-cols-for-array }
9220 {
9221     Too-much-columns.\\
9222     In~the~row~\int_eval:n { \c@iRow },~
9223     ~you~try~to~use~more~columns~than~allowed~by~your~
9224     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9225     \int_use:N \g_@@_static_num_of_col_int\
9226     ~(plus~the~potential~exterior~ones).
9227     This-error-is-fatal.
9228 }

9229 \@@_msg_new:nn { columns-not-used }
9230 {
9231     Columns-not-used.\\
9232     The-preamble-of~your~\@@_full_name_env:\ announces~\int_use:N
9233     \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9234     The-columns-you-did-not-used-won't-be-created.\\
9235     You-won't-have-similar-error-till-the-end-of-the-document.
9236 }

9237 \@@_msg_new:nn { in-first-col }
9238 {
9239     Erroneous-use.\\
9240     You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9241     That-command-will-be-ignored.
9242 }

9243 \@@_msg_new:nn { in-last-col }
9244 {
9245     Erroneous-use.\\
9246     You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9247     That-command-will-be-ignored.
9248 }

9249 \@@_msg_new:nn { in-first-row }
9250 {
9251     Erroneous-use.\\
9252     You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9253     That-command-will-be-ignored.
9254 }

9255 \@@_msg_new:nn { in-last-row }
9256 {
9257     You-can't-use-the-command~#1 in-the-last-row~(exterior)~of-the-array.\\
9258     That-command-will-be-ignored.
9259 }

9260 \@@_msg_new:nn { caption-outside-float }
9261 {
9262     Key-caption-forbidden.\\
9263     You-can't-use-the-key~'caption'~because~you-are~not~in~a~floating~
9264     environment.~This-key-will-be-ignored.
9265 }

9266 \@@_msg_new:nn { short-caption-without-caption }
9267 {
9268     You-should-not-use-the-key~'short-caption'~without~'caption'.~
9269     However,~your~'short-caption'~will~be~used~as~'caption'.
9270 }

9271 \@@_msg_new:nn { double-closing-delimiter }
9272 {
9273     Double-delimiter.\\

```

```

9274     You-can't-put-a-second-closing-delimiter-~"#1"~just-after-a-first-closing~
9275     delimiter.~This-delimiter-will-be-ignored.
9276 }
9277 \@@_msg_new:nn { delimiter-after-opening }
9278 {
9279     Double-delimiter.\\
9280     You-can't-put-a-second-delimiter-~"#1"~just-after-a-first-opening~
9281     delimiter.~That-delimiter-will-be-ignored.
9282 }
9283 \@@_msg_new:nn { bad-option-for-line-style }
9284 {
9285     Bad-line-style.\\
9286     Since-you-haven't-loaded-Tikz,~the-only-value-you-can-give-to-'line-style'~
9287     is-'standard'.~That-key-will-be-ignored.
9288 }
9289 \@@_msg_new:nn { Identical-notes-in-caption }
9290 {
9291     Identical-tabular-notes.\\
9292     You-can't-put-several-notes-with-the-same-content-in~
9293     \token_to_str:N \caption\ (but-you-can-in-the-main-tabular).\\
9294     If-you-go-on,~the-output-will-probably-be-erroneous.
9295 }
9296 \@@_msg_new:nn { tabularnote-below-the-tabular }
9297 {
9298     \token_to_str:N \tabularnote\ forbidden\\
9299     You-can't-use-\token_to_str:N \tabularnote\ in-the-caption~
9300     of-your-tabular-because-the-caption-will-be-composed-below~
9301     the-tabular.~If-you-want-the-caption-above-the-tabular-use-the~
9302     key-'caption-above'~in-\token_to_str:N \NiceMatrixOptions.\\
9303     Your-\token_to_str:N \tabularnote\ will-be-discarded-and~
9304     no-similar-error-will-be-raised-in-this-document.
9305 }
9306 \@@_msg_new:nn { Unknown-key-for-rules }
9307 {
9308     Unknown-key.\\
9309     There-is-only-two-keys-available-here:~width-and-color.\\
9310     Your-key-'~\l_keys_key_str'~will-be-ignored.
9311 }
9312 \@@_msg_new:nn { Unknown-key-for-TikzEveryCell }
9313 {
9314     Unknown-key.\\
9315     There-is-only-two-keys-available-here:~
9316     'empty'~and~'not-empty'.\\
9317     Your-key-'~\l_keys_key_str'~will-be-ignored.
9318 }
9319 \@@_msg_new:nn { Unknown-key-for-rotate }
9320 {
9321     Unknown-key.\\
9322     The-only-key-available-here-is-'c'.\\
9323     Your-key-'~\l_keys_key_str'~will-be-ignored.
9324 }
9325 \@@_msg_new:nnn { Unknown-key-for-custom-line }
9326 {
9327     Unknown-key.\\
9328     The-key-'~\l_keys_key_str'~is-unknown-in-a-'custom-line'.~
9329     It-you-go-on,~you-will-probably-have-other-errors. \\
9330     \c_@@_available_keys_str
9331 }
9332 {
9333     The-available-keys-are-(in-alphabetic-order):~

```

```

9334     ccommand,~
9335     color,~
9336     command,~
9337     dotted,~
9338     letter,~
9339     multiplicity,~
9340     sep-color,~
9341     tikz,~and~total-width.
9342 }
9343 \@@_msg_new:nnn { Unknown~key~for~xdots }
9344 {
9345     Unknown~key.\\
9346     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9347     \c_@@_available_keys_str
9348 }
9349 {
9350     The~available~keys~are~(in~alphabetic~order):~
9351     'color',~
9352     'horizontal-labels',~
9353     'inter',~
9354     'line-style',~
9355     'radius',~
9356     'shorten',~
9357     'shorten-end'~and~'shorten-start'.
9358 }
9359 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9360 {
9361     Unknown~key.\\
9362     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9363     (and~you~try~to~use~'\l_keys_key_str')\\
9364     That~key~will~be~ignored.
9365 }
9366 \@@_msg_new:nn { label~without~caption }
9367 {
9368     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9369     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9370 }
9371 \@@_msg_new:nn { W~warning }
9372 {
9373     Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9374     (row~\int_use:N \c@iRow).
9375 }
9376 \@@_msg_new:nn { Construct~too~large }
9377 {
9378     Construct~too~large.\\
9379     Your~command~\token_to_str:N #1
9380     can't~be~drawn~because~your~matrix~is~too~small.\\
9381     That~command~will~be~ignored.
9382 }
9383 \@@_msg_new:nn { underscore~after~nicematrix }
9384 {
9385     Problem~with~'underscore'.\\
9386     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9387     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9388     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9389 }
9390 \@@_msg_new:nn { ampersand~in~light-syntax }
9391 {
9392     Ampersand~forbidden.\\
9393     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9394     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.

```

```

9395 }
9396 \@@_msg_new:nn { double-backslash~in~light-syntax }
9397 {
9398   Double~backslash~forbidden.\\
9399   You~can't~use~\token_to_str:N
9400   \\~to~separate~rows~because~the~key~'light-syntax'~
9401   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9402   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9403 }
9404 \@@_msg_new:nn { hlines~with~color }
9405 {
9406   Incompatible~keys.\\
9407   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9408   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9409   Maybe~it~will~possible~in~future~version.\\
9410   Your~key~will~be~discarded.
9411 }
9412 \@@_msg_new:nn { bad~value~for~baseline }
9413 {
9414   Bad~value~for~baseline.\\
9415   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9416   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9417   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9418   the~form~'line-i'.\\
9419   A~value~of~1~will~be~used.
9420 }
9421 \@@_msg_new:nn { detection~of~empty~cells }
9422 {
9423   Problem~with~'not-empty'\\
9424   For~technical~reasons,~you~must~activate~
9425   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9426   in~order~to~use~the~key~'\l_keys_key_str'.\\
9427   That~key~will~be~ignored.
9428 }
9429 \@@_msg_new:nn { siunitx~not~loaded }
9430 {
9431   siunitx~not~loaded\\
9432   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9433   That~error~is~fatal.
9434 }
9435 \@@_msg_new:nn { ragged2e~not~loaded }
9436 {
9437   You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9438   your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9439   \l_keys_key_str'~will~be~used~instead.
9440 }
9441 \@@_msg_new:nn { Invalid~name }
9442 {
9443   Invalid~name.\\
9444   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9445   \SubMatrix\ of~your~\@@_full_name_env:.\
9446   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9447   This~key~will~be~ignored.
9448 }
9449 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9450 {
9451   Wrong~line.\\
9452   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9453   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9454   number~is~not~valid.~It~will~be~ignored.

```

```

9455 }
9456 \@@_msg_new:nn { Impossible-delimiter }
9457 {
9458   Impossible-delimiter.\\
9459   It's-impossible-to-draw-the-#1-delimiter-of-your-
9460   \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty-
9461   in-that-column.
9462   \bool_if:NT \l_@@_submatrix_slim_bool
9463   { ~Maybe-you-should-try-without-the-key-'slim'. } \\
9464   This-\token_to_str:N \SubMatrix\ will-be-ignored.
9465 }
9466 \@@_msg_new:nnn { width-without-X-columns }
9467 {
9468   You-have-used-the-key-'width'~but-you-have-put-no-'X'~column.~
9469   That-key-will-be-ignored.
9470 }
9471 {
9472   This-message-is-the-message-'width-without-X-columns'~
9473   of-the-module-'nicematrix'.~
9474   The-experimented-users-can-disable-that-message-with-
9475   \token_to_str:N \msg_redirect_name:nnn.\\
9476 }
9477
9478 \@@_msg_new:nn { key-multiplicity-with-dotted }
9479 {
9480   Incompatible-keys. \\
9481   You-have-used-the-key-'multiplicity'~with-the-key-'dotted'~
9482   in-a-'custom-line'.~They-are-incompatible. \\
9483   The-key-'multiplicity'~will-be-discarded.
9484 }
9485 \@@_msg_new:nn { empty-environment }
9486 {
9487   Empty-environment.\\
9488   Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9489 }
9490 \@@_msg_new:nn { No-letter-and-no-command }
9491 {
9492   Erroneous-use.\\
9493   Your-use-of-'custom-line'~is-no-op~since-you-don't-have-used-the-
9494   key-'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys-'command'~or~
9495   ~'ccommand'~(to-draw-horizontal-rules).\\
9496   However,~you~can~go~on.
9497 }
9498 \@@_msg_new:nn { Forbidden-letter }
9499 {
9500   Forbidden-letter.\\
9501   You-can't-use-the-letter~'#1'~for-a-customized-line.\\
9502   It~will~be~ignored.
9503 }
9504 \@@_msg_new:nn { Several-letters }
9505 {
9506   Wrong-name.\\
9507   You-must-use-only-one-letter~as-value-for-the-key-'letter'~(and-you-
9508   have-used~'\l_@@_letter_str').\\
9509   It~will~be~ignored.
9510 }
9511 \@@_msg_new:nn { Delimiter-with-small }
9512 {
9513   Delimiter~forbidden.\\
9514   You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\

```

```

9515     because~the~key~'small'~is~in~force.\\
9516     This~error~is~fatal.
9517 }
9518 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9519 {
9520     Unknown~cell.\\
9521     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9522     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9523     can't~be~executed~because~a~cell~doesn't~exist.\\
9524     This~command~\token_to_str:N \line\ will~be~ignored.
9525 }
9526 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9527 {
9528     Duplicate~name.\\
9529     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9530     in~this~\@@_full_name_env:.\\
9531     This~key~will~be~ignored.\\
9532     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9533     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9534 }
9535 {
9536     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9537     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9538 }
9539 \@@_msg_new:nn { r~or~l~with~preamble }
9540 {
9541     Erroneous~use.\\
9542     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9543     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9544     your~\@@_full_name_env:.\\
9545     This~key~will~be~ignored.
9546 }
9547 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9548 {
9549     Erroneous~use.\\
9550     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9551     the~array.~This~error~is~fatal.
9552 }
9553 \@@_msg_new:nn { bad~corner }
9554 {
9555     Bad~corner.\\
9556     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9557     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9558     This~specification~of~corner~will~be~ignored.
9559 }
9560 \@@_msg_new:nn { bad~border }
9561 {
9562     Bad~border.\\
9563     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9564     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9565     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9566     also~use~the~key~'tikz'
9567     \IfPackageLoadedTF { tikz }
9568     { }
9569     {~if~you~load~the~LaTeX~package~'tikz'}).\\
9570     This~specification~of~border~will~be~ignored.
9571 }
9572 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9573 {
9574     TikZ~not~loaded.\\
9575     You~can't~use~\token_to_str:N \TikzEveryCell\

```

```

9576     because-you-have-not-loaded-tikz.~
9577     This-command-will-be-ignored.
9578 }
9579 \@@_msg_new:nn { tikz-key-without-tikz }
9580 {
9581     TikZ-not-loaded.\\
9582     You-can't-use-the-key-'tikz'~for-the-command~'\token_to_str:N
9583     \Block'~because-you-have-not-loaded-tikz.~
9584     This-key-will-be-ignored.
9585 }
9586 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
9587 {
9588     Erroneous-use.\\
9589     In-the~\@@_full_name_env:,~you-must-use-the-key~
9590     'last-col'~without-value.\\
9591     However,~you-can-go-on-for-this-time~
9592     (the-value~'\l_keys_value_tl'~will-be-ignored).
9593 }
9594 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9595 {
9596     Erroneous-use.\\
9597     In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9598     'last-col'~without-value.\\
9599     However,~you-can-go-on-for-this-time~
9600     (the-value~'\l_keys_value_tl'~will-be-ignored).
9601 }
9602 \@@_msg_new:nn { Block-too-large-1 }
9603 {
9604     Block-too-large.\\
9605     You-try-to-draw-a-block-in-the-cell~#1-#2-of-your-matrix-but-the-matrix-is-
9606     too-small-for-that-block. \\
9607     This-block-and-maybe-others-will-be-ignored.
9608 }
9609 \@@_msg_new:nn { Block-too-large-2 }
9610 {
9611     Block-too-large.\\
9612     The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9613     \g_@@_static_num_of_col_int\
9614     columns-but-you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
9615     specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9616     (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:~\\
9617     This-block-and-maybe-others-will-be-ignored.
9618 }
9619 \@@_msg_new:nn { unknown-column-type }
9620 {
9621     Bad-column-type.\\
9622     The-column-type~'#1'~in-your~\@@_full_name_env:\
9623     is-unknown. \\
9624     This-error-is-fatal.
9625 }
9626 \@@_msg_new:nn { unknown-column-type-S }
9627 {
9628     Bad-column-type.\\
9629     The-column-type~'S'~in-your~\@@_full_name_env:\ is-unknown. \\
9630     If-you-want-to-use-the-column-type~'S'~of-siunitx,~you-should-
9631     load-that-package. \\
9632     This-error-is-fatal.
9633 }
9634 \@@_msg_new:nn { tabularnote-forbidden }
9635 {

```

```

9636     Forbidden-command.\
9637     You~can't~use~the~command~\token_to_str:N\tabularnote\
9638     ~here.~This~command~is~available~only~in~
9639     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9640     the~argument~of~a~command~\token_to_str:N \caption\ included~
9641     in~an~environment~{table}. \
9642     This~command~will~be~ignored.
9643   }

9644   \@@_msg_new:nn { borders~forbidden }
9645   {
9646     Forbidden-key.\
9647     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9648     because~the~option~'rounded-corners'~
9649     is~in~force~with~a~non-zero~value.\
9650     This~key~will~be~ignored.
9651   }

9652   \@@_msg_new:nn { bottomrule~without~booktabs }
9653   {
9654     booktabs~not~loaded.\
9655     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9656     loaded~'booktabs'.\
9657     This~key~will~be~ignored.
9658   }

9659   \@@_msg_new:nn { enumitem~not~loaded }
9660   {
9661     enumitem~not~loaded.\
9662     You~can't~use~the~command~\token_to_str:N\tabularnote\
9663     ~because~you~haven't~loaded~'enumitem'.\
9664     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9665     ignored~in~the~document.
9666   }

9667   \@@_msg_new:nn { tikz~without~tikz }
9668   {
9669     Tikz~not~loaded.\
9670     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9671     loaded.~If~you~go~on,~that~key~will~be~ignored.
9672   }

9673   \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9674   {
9675     Tikz~not~loaded.\
9676     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9677     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9678     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9679     use~that~custom~line.
9680   }

9681   \@@_msg_new:nn { tikz~in~borders~without~tikz }
9682   {
9683     Tikz~not~loaded.\
9684     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9685     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9686     That~key~will~be~ignored.
9687   }

9688   \@@_msg_new:nn { without~color~inside }
9689   {
9690     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9691     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9692     outside~\token_to_str:N \CodeBefore,~you~
9693     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\
9694     You~can~go~on~but~you~may~need~more~compilations.
9695   }

```

```

9696 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9697 {
9698   Erroneous-use.\
9699   In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
9700   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9701   The~key~'color'~will~be~discarded.
9702 }
9703 \@@_msg_new:nn { Wrong-last-row }
9704 {
9705   Wrong-number.\
9706   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9707   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9708   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9709   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9710   without~value~(more~compilations~might~be~necessary).
9711 }
9712 \@@_msg_new:nn { Yet-in-env }
9713 {
9714   Nested-environments.\
9715   Environments-of~nicematrix~can't~be~nested.\
9716   This~error~is~fatal.
9717 }
9718 \@@_msg_new:nn { Outside-math-mode }
9719 {
9720   Outside-math-mode.\
9721   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9722   (and~not~in~\token_to_str:N \vcenter).\
9723   This~error~is~fatal.
9724 }
9725 \@@_msg_new:nn { One-letter-allowed }
9726 {
9727   Bad-name.\
9728   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
9729   It~will~be~ignored.
9730 }
9731 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9732 {
9733   Environment~{TabularNote}~forbidden.\
9734   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9735   but~*before*~the~\token_to_str:N \CodeAfter.\
9736   This~environment~{TabularNote}~will~be~ignored.
9737 }
9738 \@@_msg_new:nn { varwidth-not-loaded }
9739 {
9740   varwidth-not-loaded.\
9741   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9742   loaded.\
9743   Your~column~will~behave~like~'p'.
9744 }
9745 \@@_msg_new:nnn { Unknow-key-for-RulesBis }
9746 {
9747   Unknow-key.\
9748   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
9749   \c_@@_available_keys_str
9750 }
9751 {
9752   The~available~keys~are~(in~alphabetic~order):~
9753   color,~
9754   dotted,~
9755   multiplicity,~
9756   sep-color,~

```

```

9757     tikz,~and~total~width.
9758 }
9759
9760 \@@_msg_new:nnn { Unknown~key~for~Block }
9761 {
9762     Unknown~key.\\
9763     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9764     \Block.\\ It~will~be~ignored. \\
9765     \c_@@_available_keys_str
9766 }
9767 {
9768     The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9769     hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9770     respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9771 }
9772 \@@_msg_new:nnn { Unknown~key~for~Brace }
9773 {
9774     Unknown~key.\\
9775     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9776     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9777     It~will~be~ignored. \\
9778     \c_@@_available_keys_str
9779 }
9780 {
9781     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9782     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9783     right~shorten)~and~yshift.
9784 }
9785 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9786 {
9787     Unknown~key.\\
9788     The~key~'\l_keys_key_str'~is~unknown.\\
9789     It~will~be~ignored. \\
9790     \c_@@_available_keys_str
9791 }
9792 {
9793     The~available~keys~are~(in~alphabetic~order):~
9794     delimiters/color,~
9795     rules~(with~the~subkeys~'color'~and~'width'),~
9796     sub~matrix~(several~subkeys)~
9797     and~xdots~(several~subkeys).~
9798     The~latter~is~for~the~command~\token_to_str:N \line.
9799 }
9800 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9801 {
9802     Unknown~key.\\
9803     The~key~'\l_keys_key_str'~is~unknown.\\
9804     It~will~be~ignored. \\
9805     \c_@@_available_keys_str
9806 }
9807 {
9808     The~available~keys~are~(in~alphabetic~order):~
9809     create~cell~nodes,~
9810     delimiters/color~and~
9811     sub~matrix~(several~subkeys).
9812 }
9813 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9814 {
9815     Unknown~key.\\
9816     The~key~'\l_keys_key_str'~is~unknown.\\
9817     That~key~will~be~ignored. \\
9818     \c_@@_available_keys_str

```

```

9819 }
9820 {
9821   The-available-keys-are~(in~alphabetic~order):~
9822   'delimiters/color',~
9823   'extra-height',~
9824   'hlines',~
9825   'hvlines',~
9826   'left-xshift',~
9827   'name',~
9828   'right-xshift',~
9829   'rules'~(with~the~subkeys~'color'~and~'width'),~
9830   'slim',~
9831   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9832   and~'right-xshift').\\
9833 }
9834 \@@_msg_new:nnn { Unknown~key~for~notes }
9835 {
9836   Unknown-key.\\
9837   The-key~'\l_keys_key_str'~is~unknown.\\
9838   That~key~will~be~ignored. \\
9839   \c_@@_available_keys_str
9840 }
9841 {
9842   The-available-keys-are~(in~alphabetic~order):~
9843   bottomrule,~
9844   code-after,~
9845   code-before,~
9846   detect-duplicates,~
9847   enumitem-keys,~
9848   enumitem-keys-para,~
9849   para,~
9850   label-in-list,~
9851   label-in-tabular~and~
9852   style.
9853 }
9854 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9855 {
9856   Unknown-key.\\
9857   The-key~'\l_keys_key_str'~is~unknown~for~the~command~
9858   \token_to_str:N \RowStyle. \\
9859   That~key~will~be~ignored. \\
9860   \c_@@_available_keys_str
9861 }
9862 {
9863   The-available-keys-are~(in~alphabetic~order):~
9864   'bold',~
9865   'cell-space-top-limit',~
9866   'cell-space-bottom-limit',~
9867   'cell-space-limits',~
9868   'color',~
9869   'nb-rows'~and~
9870   'rowcolor'.
9871 }
9872 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9873 {
9874   Unknown-key.\\
9875   The-key~'\l_keys_key_str'~is~unknown~for~the~command~
9876   \token_to_str:N \NiceMatrixOptions. \\
9877   That~key~will~be~ignored. \\
9878   \c_@@_available_keys_str
9879 }
9880 {
9881   The-available-keys-are~(in~alphabetic~order):~

```

```

9882 allow-duplicate-names,~
9883 caption-above,~
9884 cell-space-bottom-limit,~
9885 cell-space-limits,~
9886 cell-space-top-limit,~
9887 code-for-first-col,~
9888 code-for-first-row,~
9889 code-for-last-col,~
9890 code-for-last-row,~
9891 corners,~
9892 custom-key,~
9893 create-extra-nodes,~
9894 create-medium-nodes,~
9895 create-large-nodes,~
9896 delimiters~(several~subkeys),~
9897 end-of-row,~
9898 first-col,~
9899 first-row,~
9900 hlines,~
9901 hvlines,~
9902 hvlines-except-borders,~
9903 last-col,~
9904 last-row,~
9905 left-margin,~
9906 light-syntax,~
9907 matrix/columns-type,~
9908 no-cell-nodes,~
9909 notes~(several~subkeys),~
9910 nullify-dots,~
9911 pgf-node-code,~
9912 renew-dots,~
9913 renew-matrix,~
9914 respect-arraystretch,~
9915 rounded-corners,~
9916 right-margin,~
9917 rules~(with~the~subkeys~'color'~and~'width'),~
9918 small,~
9919 sub-matrix~(several~subkeys),~
9920 vlines,~
9921 xdots~(several~subkeys).
9922 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9923 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9924 {
9925   Unknown~key.\\
9926   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9927   \{NiceArray\}. \\
9928   That~key~will~be~ignored. \\
9929   \c_@@_available_keys_str
9930 }
9931 {
9932   The~available~keys~are~(in~alphabetic~order):~
9933   b,~
9934   baseline,~
9935   c,~
9936   cell-space-bottom-limit,~
9937   cell-space-limits,~
9938   cell-space-top-limit,~
9939   code-after,~
9940   code-for-first-col,~
9941   code-for-first-row,~
9942   code-for-last-col,~

```

```

9943 code-for-last-row,~
9944 color-inside,~
9945 columns-width,~
9946 corners,~
9947 create-extra-nodes,~
9948 create-medium-nodes,~
9949 create-large-nodes,~
9950 extra-left-margin,~
9951 extra-right-margin,~
9952 first-col,~
9953 first-row,~
9954 hlines,~
9955 hvlines,~
9956 hvlines-except-borders,~
9957 last-col,~
9958 last-row,~
9959 left-margin,~
9960 light-syntax,~
9961 name,~
9962 no-cell-nodes,~
9963 nullify-dots,~
9964 pgf-node-code,~
9965 renew-dots,~
9966 respect-arraystretch,~
9967 right-margin,~
9968 rounded-corners,~
9969 rules~(with~the~subkeys~'color'~and~'width'),~
9970 small,~
9971 t,~
9972 vlines,~
9973 xdots/color,~
9974 xdots/shorten-start,~
9975 xdots/shorten-end,~
9976 xdots/shorten-and~
9977 xdots/line-style.
9978 }

```

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).

```

9979 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9980 {
9981   Unknown~key.\\
9982   The~key~'\l_keys_key_str'~is~unknown~for~the~
9983   \@@_full_name_env:. \\
9984   That~key~will~be~ignored. \\
9985   \c_@@_available_keys_str
9986 }
9987 {
9988   The~available~keys~are~(in~alphabetic~order):~
9989   b,~
9990   baseline,~
9991   c,~
9992   cell-space-bottom-limit,~
9993   cell-space-limits,~
9994   cell-space-top-limit,~
9995   code-after,~
9996   code-for-first-col,~
9997   code-for-first-row,~
9998   code-for-last-col,~
9999   code-for-last-row,~
10000 color-inside,~
10001 columns-type,~
10002 columns-width,~
10003 corners,~

```

```

10004     create-extra-nodes,~
10005     create-medium-nodes,~
10006     create-large-nodes,~
10007     extra-left-margin,~
10008     extra-right-margin,~
10009     first-col,~
10010     first-row,~
10011     hlines,~
10012     hvlines,~
10013     hvlines-except-borders,~
10014     l,~
10015     last-col,~
10016     last-row,~
10017     left-margin,~
10018     light-syntax,~
10019     name,~
10020     no-cell-nodes,~
10021     nullify-dots,~
10022     pgf-node-code,~
10023     r,~
10024     renew-dots,~
10025     respect-arraystretch,~
10026     right-margin,~
10027     rounded-corners,~
10028     rules~(with~the~subkeys~'color'~and~'width'),~
10029     small,~
10030     t,~
10031     vlines,~
10032     xdots/color,~
10033     xdots/shorten-start,~
10034     xdots/shorten-end,~
10035     xdots/shorten-and~
10036     xdots/line-style.
10037 }
10038 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10039 {
10040     Unknown~key.\\
10041     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10042     \{NiceTabular\}. \\
10043     That~key~will~be~ignored. \\
10044     \c_@@_available_keys_str
10045 }
10046 {
10047     The~available~keys~are~(in~alphabetic~order):~
10048     b,~
10049     baseline,~
10050     c,~
10051     caption,~
10052     cell-space-bottom-limit,~
10053     cell-space-limits,~
10054     cell-space-top-limit,~
10055     code-after,~
10056     code-for-first-col,~
10057     code-for-first-row,~
10058     code-for-last-col,~
10059     code-for-last-row,~
10060     color-inside,~
10061     columns-width,~
10062     corners,~
10063     custom-line,~
10064     create-extra-nodes,~
10065     create-medium-nodes,~
10066     create-large-nodes,~

```

```

10067     extra-left-margin,~
10068     extra-right-margin,~
10069     first-col,~
10070     first-row,~
10071     hlines,~
10072     hvlines,~
10073     hvlines-except-borders,~
10074     label,~
10075     last-col,~
10076     last-row,~
10077     left-margin,~
10078     light-syntax,~
10079     name,~
10080     no-cell-nodes,~
10081     notes~(several~subkeys),~
10082     nullify-dots,~
10083     pgf-node-code,~
10084     renew-dots,~
10085     respect-arraystretch,~
10086     right-margin,~
10087     rounded-corners,~
10088     rules~(with~the~subkeys~'color'~and~'width'),~
10089     short-caption,~
10090     t,~
10091     tabularnote,~
10092     vlines,~
10093     xdots/color,~
10094     xdots/shorten-start,~
10095     xdots/shorten-end,~
10096     xdots/shorten-and~
10097     xdots/line-style.
10098 }

10099 \@@_msg_new:nnn { Duplicate-name }
10100 {
10101   Duplicate-name.\
10102   The-name-'l_keys_value_tl'-is-already-used-and-you-shouldn't-use~
10103   the-same-environment-name-twice.~You-can-go-on,~but,~
10104   maybe,~you-will-have-incorrect-results-especially~
10105   if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
10106   message-again,~use-the-key~'allow-duplicate-names'~in~
10107   '\token_to_str:N \NiceMatrixOptions'.\
10108   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10109   { For-a-list-of-the-names-already-used,~type-H~<return>. }
10110 }
10111 {
10112   The-names-already-defined-in-this-document-are:-
10113   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10114 }

10115 \@@_msg_new:nn { Option-auto-for-columns-width }
10116 {
10117   Erroneous-use.\
10118   You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
10119   That-key-will-be-ignored.
10120 }

10121 \@@_msg_new:nn { NiceTabularX-without-X }
10122 {
10123   NiceTabularX-without-X.\
10124   You-should-not-use~{NiceTabularX}~without-X-columns.\
10125   However,~you-can-go-on.
10126 }

10127 \@@_msg_new:nn { Preamble-forgotten }
10128 {

```

```
10129     Preamble~forgotten.\\
10130     You~have~probably~forgotten~the~preamble~of~your~
10131     \\@@_full_name_env:. \\
10132     This~error~is~fatal.
10133 }
```

# Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
3	Collecting options	4
4	Technical definitions	4
5	Parameters	10
6	The command <code>\tabularnote</code>	20
7	Command for creation of rectangle nodes	25
8	The options	26
9	Important code used by <code>{NiceArrayWithDelims}</code>	36
10	The <code>\CodeBefore</code>	50
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	58
13	The redefinition of <code>\multicolumn</code>	73
14	The environment <code>{NiceMatrix}</code> and its variants	91
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
16	After the construction of the array	93
17	We draw the dotted lines	100
18	The actual instructions for drawing the dotted lines with Tikz	113
19	User commands available in the new environments	118
20	The command <code>\line</code> accessible in code-after	124
21	The command <code>\RowStyle</code>	126
22	Colors of cells, rows and columns	128
23	The vertical and horizontal rules	141
24	The empty corners	155
25	The environment <code>{NiceMatrixBlock}</code>	158
26	The extra nodes	159
27	The blocks	164
28	How to draw the dotted lines transparently	183
29	Automatic arrays	184
30	The redefinition of the command <code>\dotfill</code>	185

31	The command <code>\diagbox</code>	185
32	The keyword <code>\CodeAfter</code>	187
33	The delimiters in the preamble	188
34	The command <code>\SubMatrix</code>	189
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	197
36	The command <code>TikzEveryCell</code>	200
37	The command <code>\ShowCellNames</code>	202
38	We process the options at package loading	204
39	About the package underscore	206
40	Error messages of the package	206