

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

May 25, 2023

## 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 registered for this package.

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

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

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

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

```
3 \RequirePackage{l3keys2e}
4 \ProvidesExplPackage
5   {nicematrix}
6   {\myfiledate}
7   {\myfileversion}
8   {Enhanced arrays with the help of PGF/TikZ}
```

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 6.19a of `nicematrix`, at the date of 2023/05/25.

```

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 x }
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, 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 \c_@@_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 currying.

```

25 \cs_new_protected:Npn \@@_error_or_warning:n
26 { \bool_if:NTF \c_@@_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_set:Nn \c_@@_messages_for_Overleaf_bool
28 {
29   \str_if_eq_p:Vn \c_sys_jobname_str { _region_ } % for Emacs
30   || \str_if_eq_p:Vn \c_sys_jobname_str { output } % for Overleaf
31 }

```

```

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

```

## 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.

```

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

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

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

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

```

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).

```

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

```

### 3 Technical definitions

```

95 \tl_new:N \l_@@_argspec_tl
96 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
97 \cs_generate_variant:Nn \keys_define:nn { n x }
98 \cs_generate_variant:Nn \str_lowercase:n { V }

```

```

99 \hook_gput_code:nnn { begindocument } { . }
100 {
101   \IfPackageLoadedTF { tikz }
102   {

```

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.

```

103   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
104   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
105 }
106 {
107   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
108   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
109 }
110 }

```

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

```

111 \@ifclassloaded { revtex4-1 }
112 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
113 {
114   \@ifclassloaded { revtex4-2 }
115   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
116   {

```

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

```

117   \cs_if_exist:NT \rvtx@ifformat@geq
118   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
119   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
120 }
121 }

```

```

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

```

The following regex will be used to modify the preamble of the array when the key `colortbl-like` is used.

```

123 \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }

```

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.

```

124 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
125 {
126   \iow_now:Nn \@mainaux
127   {
128     \ExplSyntaxOn

```

```

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

```

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.

```

135 \ProvideDocumentCommand \iddots { }
136 {
137   \mathinner
138   {
139     \tex_mkern:D 1 mu
140     \box_move_up:nn { 1 pt } { \hbox:n { . } }
141     \tex_mkern:D 2 mu
142     \box_move_up:nn { 4 pt } { \hbox:n { . } }
143     \tex_mkern:D 2 mu
144     \box_move_up:nn { 7 pt }
145     { \vbox:n { \kern 7 pt \hbox:n { . } } }
146     \tex_mkern:D 1 mu
147   }
148 }

```

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.

```

149 \hook_gput_code:nnn { begindocument } { . }
150 {
151   \IfPackageLoadedTF { booktabs }
152   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
153   { }
154 }
155 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
156 {
157   \cs_set_eq:NN \@_old_pgful@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`).

```

158   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
159   {
160     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
161     { \@_old_pgful@check@rerun { ##1 } { ##2 } }
162   }
163 }

```

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

```

164 \hook_gput_code:nnn { begindocument } { . }
165 {
166   \IfPackageLoadedTF { colortbl }
167   { }
168   {

```

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.

```

169     \cs_set_protected:Npn \CT@arc@ { }
170     \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
171     \cs_set:Npn \CT@arc@ #1 #2
172     {

```

```

173         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
174         { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
175     }
Idem for \CT@drs@.
176     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
177     \cs_set:Npn \CT@drs #1 #2
178     {
179         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
180         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
181     }
182     \cs_set:Npn \hline
183     {
184         \noalign { \ifnum 0 = ` } \fi
185         \cs_set_eq:NN \hskip \vskip
186         \cs_set_eq:NN \vrule \hrule
187         \cs_set_eq:NN \@width \@height
188         { \CT@arc@ \vline }
189         \futurelet \reserved@a
190         \@xhline
191     }
192 }
193 }

```

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.

```

194 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
195 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
196 {
197     \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
198     \int_compare:nNnT { #1 } > 1 { \multispan { \int_eval:n { #1 - 1 } } & }
199     \multispan { \int_eval:n { #2 - #1 + 1 } }
200     {
201         \CT@arc@
202         \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>

```

203     \skip_horizontal:N \c_zero_dim
204 }

```

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`.

```

205     \everycr { }
206     \cr
207     \noalign { \skip_vertical:N -\arrayrulewidth }
208 }

```

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.

```

209 \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`.

```

210 { \@@_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*.

```

211 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
212 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop

```

---

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

```

213 {
214   \tl_if_empty:nTF { #3 }
215     { \@@_cline_iii:w #1|#2-#2 \q_stop }
216     { \@@_cline_ii:w #1|#2-#3 \q_stop }
217   }
218   \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
219     { \@@_cline_iii:w #1|#2-#3 \q_stop }
220   \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
221     {

```

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).

```

222   \int_compare:nNnT { #1 } < { #2 }
223     { \multispan { \int_eval:n { #2 - #1 } } & }
224   \multispan { \int_eval:n { #3 - #2 + 1 } }
225     {
226       \CT@arc@
227       \leaders \hrule \@height \arrayrulewidth \hfill
228       \skip_horizontal:N \c_zero_dim
229     }

```

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

```

230   \peek_meaning_remove_ignore_spaces:NTF \cline
231     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
232     { \everycr { } \cr }
233   }
234   \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following command is a small shortcut.

```

235   \cs_new:Npn \@@_math_toggle_token:
236     { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

237   \cs_new_protected:Npn \@@_set_CT@arc@:n #1
238     {
239       \tl_if_blank:nF { #1 }
240       {
241         \tl_if_head_eq_meaning:nNTF { #1 } [
242           { \cs_set:Npn \CT@arc@ { \color #1 } }
243           { \cs_set:Npn \CT@arc@ { \color { #1 } } }
244         ]
245       }
246       \cs_generate_variant:Nn \@@_set_CT@arc@:n { V }

247   \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
248     {
249       \tl_if_head_eq_meaning:nNTF { #1 } [
250         { \cs_set:Npn \CT@drsc@ { \color #1 } }
251         { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
252       ]
253       \cs_generate_variant:Nn \@@_set_CT@drsc@:n { V }

```

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

```

254   \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
255     {
256       \tl_if_head_eq_meaning:nNTF { #2 } [
257         { #1 #2 }
258         { #1 { #2 } }
259       ]
260       \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N V }

```

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

```

261   \cs_new_protected:Npn \@@_color:n #1
262     { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }

```

```

263 \cs_generate_variant:Nn \@@_color:n { V }

264 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

```

### The column S of siunitx

The command `\@@_renew_NC@rewrite@S:` will be used in each environment of `nicematrix` in order to “rewrite” the S column in each environment.

```

265 \hook_gput_code:nnn { begindocument } { . }
266 {
267   \IfPackageLoadedTF { siunitx }
268   {
269     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
270     {
271       \renewcommand*{\NC@rewrite@S}[1] []
272       {
\@temptokena is a toks (not supported by the L3 programming layer).
273         \tl_if_empty:nTF { ##1 }
274         {
275           \@temptokena \exp_after:wN
276           { \tex_the:D \@temptokena \@@_S: }
277         }
278         {
279           \@temptokena \exp_after:wN
280           { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
281         }
282         \NC@find
283       }
284     }
285   }
286   { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
287 }

288 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
289 {
290   \tl_set_rescan:Nno
291   #1
292   {
293     \char_set_catcode_other:N >
294     \char_set_catcode_other:N <
295   }
296   #1
297 }

```

## 4 Parameters

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

```

298 \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).

```

299 \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.

```
300 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
301 { \int_use:N \g_@@_env_int }
```

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

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

The following counter will count the environments {NiceMatrixBlock}.

```
304 \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.

```
305 \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).

```
306 \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.).

```
307 \dim_new:N \l_@@_col_width_dim
308 \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.

```
309 \int_new:N \g_@@_row_total_int
310 \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).

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

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

```
312 \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. For example, a column p[1]{3cm} will provide the value l for all the cells of the column.

```
313 \str_new:N \l_@@_hpos_cell_str
314 \str_set:Nn \l_@@_hpos_cell_str { c }
```

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.

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

Idem for the mono-row blocks.

```
316 \dim_new:N \g_@@_blocks_ht_dim
317 \dim_new:N \g_@@_blocks_dp_dim
```

The following dimension will be used by the command `\Block` for the blocks with a key of vertical position equal to T or B.

```
318 \dim_new:N \l_@@_block_ysep_dim
```

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

```
319 \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`.

```
320 \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.

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

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

```
322 \bool_new:N \l_@@_notes_detect_duplicates_bool
323 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

If the user uses `{NiceArray}` or `{NiceTabular}` the flag `\g_@@_NiceArray_bool` will be raised.

```
324 \bool_new:N \g_@@_NiceArray_bool
```

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

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

```
325 \bool_new:N \l_@@_NiceTabular_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.

```
326 \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).

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

If the user uses an environment without preamble, we will raise the following flag.

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

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

```
329 \bool_new:N \g_@@_rotate_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.

```
330 \bool_new:N \l_@@_X_column_bool
```

```
331 \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 }`).

```
332 \tl_new:N \g_@@_aux_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`. However, it does *not* contain the value provided by the final user. Indeed, a transformation is done in order to have a preamble (for the package `array`) which is `nicematrix`-aware. That transformation is done with the command `\@@_set_preamble:Nn`.

```

333 \tl_new:N \l_@@_columns_type_tl
334 \hook_gput_code:nnn { begindocument } { . }
335 { \@@_set_preamble:Nn \l_@@_columns_type_tl { c } }

336 \cs_new_protected:Npn \@@_test_if_math_mode:
337 {
338   \if_mode_math: \else:
339     \@@_fatal:n { Outside-math-mode }
340   \fi:
341 }
```

The letter used for the `vlines` which will be drawn only in the sub-matrices. `vlism` stands for *vertical lines in sub-matrices*.

```

342 \tl_new:N \l_@@_letter_vlism_tl
```

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.

```

343 \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”.

```

344 \colorlet { nicematrix-last-col } { . }
345 \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`).

```

346 \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*.

```

347 \tl_new:N \g_@@_com_or_env_str
348 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

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).

```

349 \cs_new:Npn \@@_full_name_env:
350 {
351   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
352     { command \space \c_backslash_str \g_@@_name_env_str }
353     { environment \space \{ \g_@@_name_env_str \} }
354 }
```

The following token list corresponds to the option `code-after` (it’s also possible to set the value of that parameter with the keyword `\CodeAfter`). That parameter is *public*.

```

355 \tl_new:N \g_nicematrix_code_after_tl
356 \bool_new:N \l_@@_in_code_after_bool
```

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

```

357 \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.

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

The following token list has a function similar to `\g_nicematrix_code_after_tl` but it is used internally by `nicematrix`. In fact, we have to distinguish between `\g_nicematrix_code_after_tl` and `\g_@@_pre_code_after_tl` because we must take care of the order in which instructions stored in that parameters are executed.

```
359 \tl_new:N \g_@@_pre_code_after_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`.

```
360 \tl_new:N \g_nicematrix_code_before_tl
```

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

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.

```
362 \int_new:N \l_@@_old_iRow_int
```

```
363 \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).

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

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

```
365 \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.

```
366 \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.

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

```
368 \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.

```
369 \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).

```
370 \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).

```
371 \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`).

```
372 \tl_new:N \l_@@_code_before_tl
373 \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).

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

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

```
375 \dim_new:N \l_@@_x_initial_dim
376 \dim_new:N \l_@@_y_initial_dim
377 \dim_new:N \l_@@_x_final_dim
378 \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.

```
379 \dim_zero_new:N \l_@@_tmpc_dim
380 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
381 \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”.

```
382 \dim_new:N \g_@@_width_last_col_dim
383 \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.

```
384 \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!).

```
385 \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}`.

```
386 \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.

```
387 \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.

```
388 \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`).

```
389 \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.

```
390 \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).

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

```
392 \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`).

```
393 \int_new:N \l_@@_row_min_int
```

```
394 \int_new:N \l_@@_row_max_int
```

```
395 \int_new:N \l_@@_col_min_int
```

```
396 \int_new:N \l_@@_col_max_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.

```
397 \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).

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

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

```
399 \tl_new:N \l_@@_fill_tl
```

```
400 \tl_new:N \l_@@_draw_tl
```

```
401 \seq_new:N \l_@@_tikz_seq
```

```
402 \clist_new:N \l_@@_borders_clist
```

```
403 \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.

```
404 \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`.

```
405 \tl_new:N \l_@@_color_tl
```

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

```
406 \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`).

```
407 \str_new:N \l_@@_hpos_block_str
408 \str_set:Nn \l_@@_hpos_block_str { c }
409 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

For the vertical position, the possible values are `c`, `t` and `b`. Of course, it would be interesting to program a key `T` and a key `B`.

```
410 \str_new:N \l_@@_vpos_of_block_str
411 \str_set:Nn \l_@@_vpos_of_block_str { c }
```

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

```
412 \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).

```
413 \bool_new:N \l_@@_vlines_block_bool
414 \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.

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

416 \dim_new:N \l_@@_submatrix_extra_height_dim
417 \dim_new:N \l_@@_submatrix_left_xshift_dim
418 \dim_new:N \l_@@_submatrix_right_xshift_dim
419 \clist_new:N \l_@@_hlines_clist
420 \clist_new:N \l_@@_vlines_clist
421 \clist_new:N \l_@@_submatrix_hlines_clist
422 \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}`).

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

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

```
424 \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`).

```
425 \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.

```
426 \int_new:N \l_@@_first_row_int
427 \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.

```
428 \int_new:N \l_@@_first_col_int
429 \int_set:Nn \l_@@_first_col_int 1
```

- **Last row**

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

```
430 \int_new:N \l_@@_last_row_int
431 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
433 \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.

```
434 \int_new:N \l_@@_last_col_int
435 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

---

<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.



```

\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.

```

436 \bool_new:N \g_@@_last_col_found_bool

```

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

## Some utilities

```

437 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
438 {
439   \tl_set:Nn \l_tmpa_tl { #1 }
440   \tl_set:Nn \l_tmpb_tl { #2 }
441 }

```

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.

```

442 \cs_new_protected:Npn \@@_expand_clist:N #1
443 {
444   \clist_if_in:NnF #1 { all }
445   {
446     \clist_clear:N \l_tmpa_clist
447     \clist_map_inline:Nn #1
448     {
449       \tl_if_in:nnTF { ##1 } { - }
450       { \@@_cut_on_hyphen:w ##1 \q_stop }
451       {
452         \tl_set:Nn \l_tmpa_tl { ##1 }
453         \tl_set:Nn \l_tmpb_tl { ##1 }
454       }
455       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
456       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
457     }
458     \tl_set_eq:NN #1 \l_tmpa_clist
459   }
460 }

```

## 5 The command `\tabularnote`

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

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

- The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.<sup>3</sup>
- During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each composant of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabulernote}`. The first composante 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 composantes 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.

```

461 \newcounter { tabulernote }
462 \seq_new:N \g_@@_notes_seq
463 \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.

```

464 \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.

```

465 \seq_new:N \l_@@_notes_labels_seq
466 \newcounter{nicematrix_draft}
467 \cs_new_protected:Npn \@@_notes_format:n #1
468 {
469   \setcounter { nicematrix_draft } { #1 }
470   \@@_notes_style:n { nicematrix_draft }
471 }

```

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

```

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

```

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

```

473 \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`.

```

474 \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`.

```

475 \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`).

```

476 \hook_gput_code:nnn { begindocument } { . }
477 {
478   \IfPackageLoadedTF { enumitem }
479   {

```

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.

```

480   \newlist { tabularnotes } { enumerate } { 1 }
481   \setlist [ tabularnotes ]
482   {
483     topsep = 0pt ,
484     noitemsep ,
485     leftmargin = * ,
486     align = left ,
487     labelsep = 0pt ,
488     label =
489       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
490   }
491   \newlist { tabularnotes* } { enumerate* } { 1 }
492   \setlist [ tabularnotes* ]
493   {
494     afterlabel = \nobreak ,
495     itemjoin = \quad ,
496     label =
497       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
498   }

```

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`.

```

499   \NewDocumentCommand \tabularnote { o m }
500   {
501     \bool_if:nT { \cs_if_exist_p:N \@capttype || \l_@@_in_env_bool }
502     {
503       \bool_if:nTF { ! \l_@@_NiceTabular_bool && \l_@@_in_env_bool }
504       { \@@_error:n { tabularnote~forbidden } }
505       {
506         \bool_if:NTF \l_@@_in_caption_bool
507         { \@@_tabularnote_caption:nn { #1 } { #2 } }
508         { \@@_tabularnote:nn { #1 } { #2 } }
509       }
510     }
511   }
512 }
513 {
514   \NewDocumentCommand \tabularnote { o m }
515   {
516     \@@_error_or_warning:n { enumitem~not~loaded }
517     \@@_gredirect_none:n { enumitem~not~loaded }
518   }
519 }
520 }

```

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`.

```

521 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
522 {

```

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).

```

523   \int_zero:N \l_tmpa_int
524   \bool_if:NT \l_@@_notes_detect_duplicates_bool
525   {

```

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`.

```

526   \seq_map_indexed_inline:Nn \g_@@_notes_seq
527   {
528     \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
529     {
530       \int_set:Nn \l_tmpa_int { ##1 }
531       \seq_map_break:
532     }
533   }
534   \int_compare:nNnF \l_tmpa_int = \c_zero_int
535   { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
536 }
537 \int_compare:nNnT \l_tmpa_int = \c_zero_int
538 {
539   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
540   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
541 }
542 \seq_put_right:Nx \l_@@_notes_labels_seq
543 {
544   \tl_if_novalue:nTF { #1 }
545   {
546     \@@_notes_format:n
547     {
548       \int_eval:n
549       {
550         \int_compare:nNnTF \l_tmpa_int = \c_zero_int
551         \c@tabularnote
552         \l_tmpa_int
553       }
554     }
555   }
556   { #1 }
557 }
558 \peek_meaning:NF \tabularnote
559 {

```

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.

```

560   \hbox_set:Nn \l_tmpa_box
561   {

```

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

```

562   \@@_notes_label_in_tabular:n
563   {
564     \seq_use:Nnnn

```

```

565         \l_@@_notes_labels_seq { , } { , } { , }
566     }
567 }

```

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

```

568     \int_gsub:Nn \c@tabularnote { 1 }
569     \int_set_eq:NN \l_tmpa_int \c@tabularnote
570     \refstepcounter { tabularnote }
571     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
572     { \int_gincr:N \c@tabularnote }
573     \seq_clear:N \l_@@_notes_labels_seq
574     \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).

```

575     \skip_horizontal:n { \box_wd:N \l_tmpa_box }
576 }
577 }

```

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`.

```

578 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
579 {
580     \bool_if:NTF \g_@@_caption_finished_bool
581     {
582         \int_compare:nNnT
583         \c@tabularnote = \g_@@_notes_caption_int
584         { \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`!

```

585     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
586     { \@@_error:n { Identical~notes~in~caption } }
587 }
588 {

```

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

```

589     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
590     {

```

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.

```

591         \bool_gset_true:N \g_@@_caption_finished_bool
592         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
593         \int_gzero:N \c@tabularnote
594     }
595     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
596 }

```

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!

```

597     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
598     \seq_put_right:Nx \l_@@_notes_labels_seq
599     {
600         \tl_if_novalue:nTF { #1 }
601         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
602         { #1 }

```

```

603     }
604     \peek_meaning:NF \tabularnote
605     {
606         \@@_notes_label_in_tabular:n
607         { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
608         \seq_clear:N \l_@@_notes_labels_seq
609     }
610 }
611 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
612 { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }

```

## 6 Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).

`#1` is the name of the node which will be created; `#2` and `#3` are the coordinates of one of the corner of the rectangle; `#4` and `#5` are the coordinates of the opposite corner.

```

613 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
614 {
615     \begin { pgfscope }
616     \pgfset
617     {
618         % outer~sep = \c_zero_dim ,
619         inner~sep = \c_zero_dim ,
620         minimum~size = \c_zero_dim
621     }
622     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
623     \pgfnode
624     { rectangle }
625     { center }
626     {
627         \vbox_to_ht:nn
628         { \dim_abs:n { #5 - #3 } }
629         {
630             \vfill
631             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
632         }
633     }
634     { #1 }
635     { }
636     \end { pgfscope }
637 }

```

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.

```

638 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
639 {
640     \begin { pgfscope }
641     \pgfset
642     {
643         % outer~sep = \c_zero_dim ,
644         inner~sep = \c_zero_dim ,
645         minimum~size = \c_zero_dim
646     }
647     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
648     \pgfpointdiff { #3 } { #2 }
649     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
650     \pgfnode
651     { rectangle }

```

```

652     { center }
653     {
654         \vbox_to_ht:nn
655         { \dim_abs:n \l_tmpb_dim }
656         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
657     }
658     { #1 }
659     { }
660 \end { pgfscope }
661 }

```

## 7 The options

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

```

662 \tl_new:N \l_@@_caption_tl
663 \tl_new:N \l_@@_short_caption_tl
664 \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).

```

665 \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 `colortbl-like` is used, these commands are available.

```

666 \bool_new:N \l_@@_colortbl_like_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`.

```

667 \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`).

```

668 \dim_new:N \l_@@_cell_space_top_limit_dim
669 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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.

```

670 \dim_new:N \l_@@_xdots_inter_dim
671 \hook_gput_code:nnn { begindocument } { . }
672 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

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

The following dimension is the minimal distance between a node (in fact an anchor of that node) and a dotted line (we say “minimal” because, by definition, a dotted line is not a continuous line and, therefore, this distance may vary a little).

```

673 \dim_new:N \l_@@_xdots_shorten_start_dim
674 \dim_new:N \l_@@_xdots_shorten_end_dim
675 \hook_gput_code:nnn { begindocument } { . }
676 {

```

```

677 \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
678 \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
679 }

```

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

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.

```

680 \dim_new:N \l_@@_xdots_radius_dim
681 \hook_gput_code:nnn { begindocument } { . }
682 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }

```

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

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.

```

683 \tl_new:N \l_@@_xdots_line_style_tl
684 \tl_const:Nn \c_@@_standard_tl { standard }
685 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

```

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

```

686 \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).

```

687 \tl_new:N \l_@@_baseline_tl
688 \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`).

```

689 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

690 \bool_new:N \l_@@_parallelize_diags_bool
691 \bool_set_true:N \l_@@_parallelize_diags_bool

```

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

```

692 \clist_new:N \l_@@_corners_clist

```

```

693 \dim_new:N \l_@@_notes_above_space_dim
694 \hook_gput_code:nnn { begindocument } { . }
695 { \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.).

```

696 \bool_new:N \l_@@_nullify_dots_bool

```



The following flag corresponds to the key `respect-arraystretch` (that key has an effect on the blocks).

```
697 \bool_new:N \l_@@_respect_arraystretch_bool
```

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).

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

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

```
699 \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.

```
700 \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”.

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

```
702 \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).

```
703 \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`).

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

```
705 \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`.

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

```
707 \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.

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

```
709 \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 “:”.

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

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

```
711 \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.

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

```

713 \keys_define:nn { NiceMatrix / xdots }
714 {
715   line-style .code:n =
716   {
717     \bool_lazy_or:nnTF
718     { \cs_if_exist_p:N \tikzpicture }
719     { \str_if_eq_p:nn { #1 } { standard } }
720     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
721     { \@@_error:n { bad-option-for-line-style } }
722   } ,
723   line-style .value_required:n = true ,
724   color .tl_set:N = \l_@@_xdots_color_tl ,
725   color .value_required:n = true ,
726   shorten .code:n =
727   \hook_gput_code:nnn { begindocument } { . }
728   {
729     \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
730     \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
731   } ,
732   shorten-start .code:n =
733   \hook_gput_code:nnn { begindocument } { . }
734   { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
735   shorten-end .code:n =
736   \hook_gput_code:nnn { begindocument } { . }
737   { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,

```

We use a hook only by security in case revtex4-1 is used (even though it is obsolete). Idem for the following keys.

```

738   shorten .value_required:n = true ,
739   shorten-start .value_required:n = true ,
740   shorten-end .value_required:n = true ,
741   radius .code:n =
742   \hook_gput_code:nnn { begindocument } { . }
743   { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
744   radius .value_required:n = true ,
745   inter .code:n =
746   \hook_gput_code:nnn { begindocument } { . }
747   { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
748   radius .value_required:n = true ,

```

The options down and up are not documented for the final user because he should use the syntax with  $\wedge$  and  $\_$ .

```

749   down .tl_set:N = \l_@@_xdots_down_tl ,
750   up .tl_set:N = \l_@@_xdots_up_tl ,

```

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

```

751   draw-first .code:n = \prg_do_nothing: ,
752   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
753 }

```

```

754 \keys_define:nn { NiceMatrix / rules }
755 {
756   color .tl_set:N = \l_@@_rules_color_tl ,
757   color .value_required:n = true ,
758   width .dim_set:N = \arrayrulewidth ,
759   width .value_required:n = true ,
760   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
761 }

```

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

```

762 \keys_define:nn { NiceMatrix / Global }
763 {
764   custom-line .code:n = \@@_custom_line:n { #1 } ,
765   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
766   rules .value_required:n = true ,
767   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
768   standard-cline .default:n = true ,
769   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
770   cell-space-top-limit .value_required:n = true ,
771   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
772   cell-space-bottom-limit .value_required:n = true ,
773   cell-space-limits .meta:n =
774   {
775     cell-space-top-limit = #1 ,
776     cell-space-bottom-limit = #1 ,
777   } ,
778   cell-space-limits .value_required:n = true ,
779   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
780   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
781   light-syntax .default:n = true ,
782   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
783   end-of-row .value_required:n = true ,
784   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
785   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
786   last-row .int_set:N = \l_@@_last_row_int ,
787   last-row .default:n = -1 ,
788   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
789   code-for-first-col .value_required:n = true ,
790   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
791   code-for-last-col .value_required:n = true ,
792   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
793   code-for-first-row .value_required:n = true ,
794   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
795   code-for-last-row .value_required:n = true ,
796   hlines .clist_set:N = \l_@@_hlines_clist ,
797   vlines .clist_set:N = \l_@@_vlines_clist ,
798   hlines .default:n = all ,
799   vlines .default:n = all ,
800   vlines-in-sub-matrix .code:n =
801   {
802     \tl_if_single_token:nTF { #1 }
803     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
804     { \@@_error:n { One~letter~allowed } }
805   } ,
806   vlines-in-sub-matrix .value_required:n = true ,
807   hvlines .code:n =
808   {
809     \bool_set_true:N \l_@@_hvlines_bool
810     \clist_set:Nn \l_@@_vlines_clist { all }
811     \clist_set:Nn \l_@@_hlines_clist { all }
812   } ,
813   hvlines-except-borders .code:n =
814   {
815     \clist_set:Nn \l_@@_vlines_clist { all }
816     \clist_set:Nn \l_@@_hlines_clist { all }
817     \bool_set_true:N \l_@@_hvlines_bool
818     \bool_set_true:N \l_@@_except_borders_bool
819   } ,
820   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.

```

821   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,

```

```

822 renew-dots .value_forbidden:n = true ,
823 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
824 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
825 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
826 create-extra-nodes .meta:n =
827   { create-medium-nodes , create-large-nodes } ,
828 left-margin .dim_set:N = \l_@@_left_margin_dim ,
829 left-margin .default:n = \arraycolsep ,
830 right-margin .dim_set:N = \l_@@_right_margin_dim ,
831 right-margin .default:n = \arraycolsep ,
832 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
833 margin .default:n = \arraycolsep ,
834 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
835 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
836 extra-margin .meta:n =
837   { extra-left-margin = #1 , extra-right-margin = #1 } ,
838 extra-margin .value_required:n = true ,
839 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
840 respect-arraystretch .default:n = true ,
841 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
842 pgf-node-code .value_required:n = true
843 }

```

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

```

844 \keys_define:nn { NiceMatrix / Env }
845   {
846     corners .clist_set:N = \l_@@_corners_clist ,
847     corners .default:n = { NW , SW , NE , SE } ,
848     code-before .code:n =
849       {
850         \tl_if_empty:nF { #1 }
851         {
852           \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
853           \bool_set_true:N \l_@@_code_before_bool
854         }
855       } ,
856     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}`.

```

857 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
858 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
859 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
860 baseline .tl_set:N = \l_@@_baseline_tl ,
861 baseline .value_required:n = true ,
862 columns-width .code:n =
863   \tl_if_eq:nnTF { #1 } { auto }
864     { \bool_set_true:N \l_@@_auto_columns_width_bool }
865     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
866 columns-width .value_required:n = true ,
867 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.

```

868 \legacy_if:nF { measuring@ }
869   {
870     \str_set:Nn \l_tmpa_str { #1 }
871     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
872       { \@@_error:nn { Duplicate-name } { #1 } }
873       { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
874     \str_set_eq:NN \l_@@_name_str \l_tmpa_str

```

```

875     } ,
876     name .value_required:n = true ,
877     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
878     code-after .value_required:n = true ,
879     colortbl-like .code:n =
880       \bool_set_true:N \l_@@_colortbl_like_bool
881       \bool_set_true:N \l_@@_code_before_bool ,
882     colortbl-like .value_forbidden:n = true
883   }
884 \keys_define:nn { NiceMatrix / notes }
885 {
886   para .bool_set:N = \l_@@_notes_para_bool ,
887   para .default:n = true ,
888   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
889   code-before .value_required:n = true ,
890   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
891   code-after .value_required:n = true ,
892   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
893   bottomrule .default:n = true ,
894   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
895   style .value_required:n = true ,
896   label-in-tabular .code:n =
897     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
898   label-in-tabular .value_required:n = true ,
899   label-in-list .code:n =
900     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
901   label-in-list .value_required:n = true ,
902   enumitem-keys .code:n =
903     {
904       \hook_gput_code:nnn { begindocument } { . }
905       {
906         \IfPackageLoadedTF { enumitem }
907           { \setlist* [ tabularnotes ] { #1 } }
908           { }
909       }
910     } ,
911   enumitem-keys .value_required:n = true ,
912   enumitem-keys-para .code:n =
913     {
914       \hook_gput_code:nnn { begindocument } { . }
915       {
916         \IfPackageLoadedTF { enumitem }
917           { \setlist* [ tabularnotes* ] { #1 } }
918           { }
919       }
920     } ,
921   enumitem-keys-para .value_required:n = true ,
922   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
923   detect-duplicates .default:n = true ,
924   unknown .code:n = \@@_error:n { Unknown-key-for-notes }
925 }
926 \keys_define:nn { NiceMatrix / delimiters }
927 {
928   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
929   max-width .default:n = true ,
930   color .tl_set:N = \l_@@_delimiters_color_tl ,
931   color .value_required:n = true ,
932 }

```

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

```

933 \keys_define:nn { NiceMatrix }

```

```

934 {
935   NiceMatrixOptions .inherit:n =
936     { NiceMatrix / Global } ,
937   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
938   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
939   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
940   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
941   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
942   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
943   CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
944   NiceMatrix .inherit:n =
945     {
946       NiceMatrix / Global ,
947       NiceMatrix / Env ,
948     } ,
949   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
950   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
951   NiceTabular .inherit:n =
952     {
953       NiceMatrix / Global ,
954       NiceMatrix / Env
955     } ,
956   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
957   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
958   NiceTabular / notes .inherit:n = NiceMatrix / notes ,
959   NiceArray .inherit:n =
960     {
961       NiceMatrix / Global ,
962       NiceMatrix / Env ,
963     } ,
964   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
965   NiceArray / rules .inherit:n = NiceMatrix / rules ,
966   pNiceArray .inherit:n =
967     {
968       NiceMatrix / Global ,
969       NiceMatrix / Env ,
970     } ,
971   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
972   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
973 }

```

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

```

974 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
975 {
976   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
977   delimiters / color .value_required:n = true ,
978   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
979   delimiters / max-width .default:n = true ,
980   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
981   delimiters .value_required:n = true ,
982   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
983   width .value_required:n = true ,
984   last-col .code:n =
985     \tl_if_empty:nF { #1 }
986     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
987     \int_zero:N \l_@@_last_col_int ,
988   small .bool_set:N = \l_@@_small_bool ,
989   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.

```

990   renew-matrix .code:n = \@@_renew_matrix: ,
991   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}`.

```
992   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.

```
993   columns-width .code:n =
994     \tl_if_eq:nnTF { #1 } { auto }
995     { \@@_error:n { Option~auto~for~columns-width } }
996     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

```
997   allow-duplicate-names .code:n =
998     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
999   allow-duplicate-names .value_forbidden:n = true ,
1000   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1001   notes .value_required:n = true ,
1002   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1003   sub-matrix .value_required:n = true ,
1004   matrix / columns-type .code:n =
1005     \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1006   matrix / columns-type .value_required:n = true ,
1007   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1008   caption-above .default:n = true ,
1009   unknown .code:n = \@@_error:n { Unknown-key~for~NiceMatrixOptions }
1010 }
```

`\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.

```
1011 \NewDocumentCommand \NiceMatrixOptions { m }
1012 { \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.

```
1013 \keys_define:nn { NiceMatrix / NiceMatrix }
1014 {
1015   last-col .code:n = \tl_if_empty:nTF {#1}
1016     {
1017       \bool_set_true:N \l_@@_last_col_without_value_bool
1018       \int_set:Nn \l_@@_last_col_int { -1 }
1019     }
1020     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1021   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1022   columns-type .value_required:n = true ,
1023   l .meta:n = { columns-type = l } ,
1024   r .meta:n = { columns-type = r } ,
1025   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1026   delimiters / color .value_required:n = true ,
1027   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1028   delimiters / max-width .default:n = true ,
1029   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1030   delimiters .value_required:n = true ,
1031   small .bool_set:N = \l_@@_small_bool ,
1032   small .value_forbidden:n = true ,
1033   unknown .code:n = \@@_error:n { Unknown-key~for~NiceMatrix }
1034 }
```

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

```
1035 \keys_define:nn { NiceMatrix / NiceArray }
1036 {
```

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.

```
1037     small .bool_set:N = \l_@@_small_bool ,
1038     small .value_forbidden:n = true ,
1039     last-col .code:n = \tl_if_empty:nF { #1 }
1040         { \@@_error:n { last-col~non-empty~for~NiceArray } }
1041         \int_zero:N \l_@@_last_col_int ,
1042     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1043     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1044     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1045 }

1046 \keys_define:nn { NiceMatrix / pNiceArray }
1047 {
1048     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1049     last-col .code:n = \tl_if_empty:nF {#1}
1050         { \@@_error:n { last-col~non-empty~for~NiceArray } }
1051         \int_zero:N \l_@@_last_col_int ,
1052     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1053     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1054     delimiters / color .value_required:n = true ,
1055     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1056     delimiters / max-width .default:n = true ,
1057     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1058     delimiters .value_required:n = true ,
1059     small .bool_set:N = \l_@@_small_bool ,
1060     small .value_forbidden:n = true ,
1061     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1062     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1063     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1064 }
```

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

```
1065 \keys_define:nn { NiceMatrix / NiceTabular }
1066 {
```

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.

```
1067     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1068         \bool_set_true:N \l_@@_width_used_bool ,
1069     width .value_required:n = true ,
1070     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
1071     rounded-corners .default:n = 4 pt ,
1072     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1073     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1074     tabularnote .value_required:n = true ,
1075     caption .tl_set:N = \l_@@_caption_tl ,
1076     caption .value_required:n = true ,
1077     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1078     short-caption .value_required:n = true ,
1079     label .tl_set:N = \l_@@_label_tl ,
1080     label .value_required:n = true ,
1081     last-col .code:n = \tl_if_empty:nF {#1}
1082         { \@@_error:n { last-col~non-empty~for~NiceArray } }
1083         \int_zero:N \l_@@_last_col_int ,
1084     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1085     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
```



```

1086     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1087 }

```

## 8 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}`).

```

1088 \cs_new_protected:Npn \@@_cell_begin:w
1089 {

```

`\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.

```

1090     \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).

```

1091     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

We increment `\c@jCol`, which is the counter of the columns.

```

1092     \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.

```

1093     \int_compare:nNnT \c@jCol = 1
1094     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_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` will be in the `\@@_cell_end:` (and the potential `\c_math_toggle_token` also).

```

1095     \hbox_set:Nw \l_@@_cell_box
1096     \bool_if:NF \l_@@_NiceTabular_bool
1097     {
1098         \c_math_toggle_token
1099         \bool_if:NT \l_@@_small_bool \scriptstyle
1100     }
1101     \g_@@_row_style_tl

```

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).

The codes `\l_@@_code_for_first_row_tl` and *al* don't apply in the corners of the matrix.

```

1102     \int_compare:nNnTF \c@iRow = 0
1103     {
1104         \int_compare:nNnT \c@jCol > 0
1105         {
1106             \l_@@_code_for_first_row_tl
1107             \xglobal \colorlet { nicematrix-first-row } { . }
1108         }
1109     }
1110     {
1111         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1112         {
1113             \l_@@_code_for_last_row_tl
1114             \xglobal \colorlet { nicematrix-last-row } { . }
1115         }
1116     }
1117 }

```

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.

```

1118 \cs_new_protected:Npn \@@_begin_of_row:
1119 {
1120   \int_gincr:N \c@iRow
1121   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1122   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1123   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1124   \pgfpicture
1125   \pgfrememberpicturepositiononpagetrue
1126   \pgfcoordinate
1127   { \@@_env: - row - \int_use:N \c@iRow - base }
1128   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1129   \str_if_empty:NF \l_@@_name_str
1130   {
1131     \pgfnodealias
1132     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1133     { \@@_env: - row - \int_use:N \c@iRow - base }
1134   }
1135   \endpgfpicture
1136 }

```

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.

```

1137 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1138 {
1139   \int_compare:nNnTF \c@iRow = 0
1140   {
1141     \dim_gset:Nn \g_@@_dp_row_zero_dim
1142     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1143     \dim_gset:Nn \g_@@_ht_row_zero_dim
1144     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1145   }
1146   {
1147     \int_compare:nNnT \c@iRow = 1
1148     {
1149       \dim_gset:Nn \g_@@_ht_row_one_dim
1150       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1151     }
1152   }
1153 }
1154 \cs_new_protected:Npn \@@_rotate_cell_box:
1155 {
1156   \box_rotate:Nn \l_@@_cell_box { 90 }
1157   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1158   {
1159     \vbox_set_top:Nn \l_@@_cell_box
1160     {
1161       \vbox_to_zero:n { }
1162       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1163       \box_use:N \l_@@_cell_box
1164     }
1165   }
1166   \bool_gset_false:N \g_@@_rotate_bool
1167 }
1168 \cs_new_protected:Npn \@@_adjust_size_box:
1169 {
1170   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim

```

```

1171 {
1172   \box_set_wd:Nn \l_@@_cell_box
1173   { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1174   \dim_gzero:N \g_@@_blocks_wd_dim
1175 }
1176 \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1177 {
1178   \box_set_dp:Nn \l_@@_cell_box
1179   { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1180   \dim_gzero:N \g_@@_blocks_dp_dim
1181 }
1182 \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1183 {
1184   \box_set_ht:Nn \l_@@_cell_box
1185   { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1186   \dim_gzero:N \g_@@_blocks_ht_dim
1187 }
1188 }
1189 \cs_new_protected:Npn \@@_cell_end:
1190 {
1191   \@@_math_toggle_token:
1192   \hbox_set_end:
1193   \@@_cell_end_i:
1194 }
1195 \cs_new_protected:Npn \@@_cell_end_i:
1196 {

```

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.

```

1197   \g_@@_cell_after_hook_tl
1198   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1199   \@@_adjust_size_box:
1200   \box_set_ht:Nn \l_@@_cell_box
1201   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1202   \box_set_dp:Nn \l_@@_cell_box
1203   { \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”).

```

1204   \dim_gset:Nn \g_@@_max_cell_width_dim
1205   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }

```

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

```

1206   \@@_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.

```

1207 \bool_if:NTF \g_@@_empty_cell_bool
1208 { \box_use_drop:N \l_@@_cell_box }
1209 {
1210   \bool_lazy_or:nnTF
1211     \g_@@_not_empty_cell_bool
1212     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1213     \@@_node_for_cell:
1214     { \box_use_drop:N \l_@@_cell_box }
1215   }
1216   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1217   \bool_gset_false:N \g_@@_empty_cell_bool
1218   \bool_gset_false:N \g_@@_not_empty_cell_bool
1219 }

```

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`).

```

1220 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1221 {
1222   \@@_math_toggle_token:
1223   \hbox_set_end:
1224   \bool_if:NF \g_@@_rotate_bool
1225   {
1226     \hbox_set:Nn \l_@@_cell_box
1227     {
1228       \makebox [ \l_@@_col_width_dim ] [ s ]
1229       { \hbox_unpack_drop:N \l_@@_cell_box }
1230     }
1231   }
1232   \@@_cell_end_i:
1233 }

```

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

```

1234 \pgfset
1235 {
1236   nicematrix / cell-node /.style =
1237   {
1238     inner~sep = \c_zero_dim ,
1239     minimum~width = \c_zero_dim
1240   }
1241 }
1242 \cs_new_protected:Npn \@@_node_for_cell:
1243 {
1244   \pgfpicture
1245   \pgfsetbaseline \c_zero_dim
1246   \pgfrememberpicturepositiononpagetrue
1247   \pgfset { nicematrix / cell-node }
1248   \pgfnode
1249   { rectangle }
1250   { base }
1251   {

```

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

```

1252 \set@color
1253 \box_use_drop:N \l_@@_cell_box

```

```

1254     }
1255     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1256     { \l_@@_pgf_node_code_tl }
1257     \str_if_empty:NF \l_@@_name_str
1258     {
1259         \pgfnodealias
1260         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1261         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1262     }
1263     \endpgfpicture
1264 }

```

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.

```

1265 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1266 {
1267     \cs_new_protected:Npn \@@_patch_node_for_cell:
1268     {
1269         \hbox_set:Nn \l_@@_cell_box
1270         {
1271             \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1272             \hbox_overlap_left:n
1273             {
1274                 \pgfsys@markposition
1275                 { \@@_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.

```

1276             #1
1277         }
1278         \box_use:N \l_@@_cell_box
1279         \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1280         \hbox_overlap_left:n
1281         {
1282             \pgfsys@markposition
1283             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1284             #1
1285         }
1286     }
1287 }
1288 }

```

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

```

1289 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1290 {
1291     \@@_patch_node_for_cell:n
1292     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1293 }
1294 { \@@_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.

```
1295 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3  
1296 {  
1297   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx  
1298   { g_@@_ #2 _ lines _ tl }  
1299   {  
1300     \use:c { @@ _ draw _ #2 : nnn }  
1301     { \int_use:N \c@iRow }  
1302     { \int_use:N \c@jCol }  
1303     { \exp_not:n { #3 } }  
1304   }  
1305 }  
1306 \cs_new_protected:Npn \@@_array:n  
1307 {  
1308   \bool_if:NTF \l_@@_NiceTabular_bool  
1309   { \dim_set_eq:NN \col@sep \tabcolsep }  
1310   { \dim_set_eq:NN \col@sep \arraycolsep }  
1311   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim  
1312   { \cs_set_nopar:Npn \@halignto { } }  
1313   { \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`.

```
1314   \@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 you need something fully expandable here.  
1315   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]  
1316 }  
1317 \cs_generate_variant:Nn \@@_array:n { V }
```

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.

```
1318 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1319 \cs_new_protected:Npn \@@_create_row_node:  
1320 {  
1321   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int  
1322   {  
1323     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow  
1324     \@@_create_row_node_i:  
1325   }  
1326 }  
  
1327 \cs_new_protected:Npn \@@_create_row_node_i:  
1328 {
```

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

```
1329   \hbox  
1330   {  
1331     \bool_if:NT \l_@@_code_before_bool  
1332     {  
1333       \vtop  
1334       {  
1335         \skip_vertical:N 0.5\arrayrulewidth  
1336         \pgfsys@markposition  
1337         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }  
1338         \skip_vertical:N -0.5\arrayrulewidth
```

```

1339     }
1340   }
1341   \pgfpicture
1342   \pgfrememberpicturepositiononpagetrue
1343   \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1344   { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1345   \str_if_empty:NF \l_@@_name_str
1346   {
1347     \pgfnodealias
1348     { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1349     { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1350   }
1351   \endpgfpicture
1352 }
1353 }

```

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

```

1354 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1355 \cs_new_protected:Npn \@@_everycr_i:
1356 {
1357   \int_gzero:N \c@jCol
1358   \bool_gset_false:N \g_@@_after_col_zero_bool
1359   \bool_if:NF \g_@@_row_of_col_done_bool
1360   {
1361     \@@_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).

```

1362   \tl_if_empty:NF \l_@@_hlines_clist
1363   {
1364     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1365     {
1366       \exp_args:NNx
1367       \clist_if_in:NnT
1368       \l_@@_hlines_clist
1369       { \int_eval:n { \c@iRow + 1 } }
1370     }
1371   }

```

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.

```

1372   \int_compare:nNnT \c@iRow > { -1 }
1373   {
1374     \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@`.

```

1375       { \hrule height \arrayrulewidth width \c_zero_dim }
1376     }
1377   }
1378 }
1379 }
1380 }

```

The command `\@@_newcolumntype` is the command `\newcolumntype` of `array` without the warnings for redefinitions of columns types (we will use it to redefine the columns types `w` and `W`).

```

1381 \cs_set_protected:Npn \@@_newcolumntype #1
1382 {
1383   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1384   \peek_meaning:NTF [
1385     { \newcol@ #1 }
1386     { \newcol@ #1 [ 0 ] }
1387   }

```

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

```

1388 \cs_set_protected:Npn \@@_renew_dots:
1389 {
1390   \cs_set_eq:NN \ldots \@@_Ldots
1391   \cs_set_eq:NN \cdots \@@_Cdots
1392   \cs_set_eq:NN \vdots \@@_Vdots
1393   \cs_set_eq:NN \ddots \@@_Ddots
1394   \cs_set_eq:NN \iddots \@@_Iddots
1395   \cs_set_eq:NN \dots \@@_Ldots
1396   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1397 }

```

When the key `colortbl-like` is used, the following code will be executed.

```

1398 \cs_new_protected:Npn \@@_colortbl_like:
1399 {
1400   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1401   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1402   \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1403 }

```

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

```

1404 \cs_new_protected:Npn \@@_pre_array_ii:
1405 {

```

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

```

1406   \int_gzero:N \g_@@_total_X_weight_int
1407   \@@_expand_clist:N \l_@@_hlines_clist
1408   \@@_expand_clist:N \l_@@_vlines_clist

```

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>.

```

1409   \IfPackageLoadedTF { booktabs }
1410   { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1411   { }
1412   \box_clear_new:N \l_@@_cell_box
1413   \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}`).

```

1414   \bool_if:NT \l_@@_small_bool
1415   {
1416     \cs_set_nopar:Npn \arraystretch { 0.47 }
1417     \dim_set:Nn \arraycolsep { 1.45 pt }
1418   }

1419   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1420   {
1421     \tl_put_right:Nn \@@_begin_of_row:
1422     {
1423       \pgfsys@markposition

```

---

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



```

1424         { \@@_env: - row - \int_use:N \c@iRow - base }
1425     }
1426 }

```

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`.

```

1427 \cs_set_nopar:Npn \ialign
1428 {
1429     \IfPackageLoadedTF { colortbl }
1430     {
1431         \CT@everycr
1432         {
1433             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1434             \@@_everycr:
1435         }
1436     }
1437     { \everycr { \@@_everycr: } }
1438     \tabskip = \c_zero_skip

```

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`.

```

1439     \dim_gzero_new:N \g_@@_dp_row_zero_dim
1440     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1441     \dim_gzero_new:N \g_@@_ht_row_zero_dim
1442     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1443     \dim_gzero_new:N \g_@@_ht_row_one_dim
1444     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1445     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1446     \dim_gzero_new:N \g_@@_ht_last_row_dim
1447     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1448     \dim_gzero_new:N \g_@@_dp_last_row_dim
1449     \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`.

```

1450     \cs_set_eq:NN \ialign \@@_old_ialign:
1451     \halign
1452 }

```

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).

```

1453     \cs_set_eq:NN \@@_old_ldots \ldots
1454     \cs_set_eq:NN \@@_old_cdots \cdots
1455     \cs_set_eq:NN \@@_old_vdots \vdots
1456     \cs_set_eq:NN \@@_old_ddots \ddots
1457     \cs_set_eq:NN \@@_old_iddots \iddots
1458     \bool_if:NTF \l_@@_standard_cline_bool
1459     { \cs_set_eq:NN \cline \@@_standard_cline }
1460     { \cs_set_eq:NN \cline \@@_cline }
1461     \cs_set_eq:NN \Ldots \@@_Ldots
1462     \cs_set_eq:NN \Cdots \@@_Cdots
1463     \cs_set_eq:NN \Vdots \@@_Vdots
1464     \cs_set_eq:NN \Ddots \@@_Ddots

```

---

<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}`.

```

1465 \cs_set_eq:NN \Iddots \@@_Iddots
1466 \cs_set_eq:NN \Hline \@@_Hline:
1467 \cs_set_eq:NN \Hspace \@@_Hspace:
1468 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1469 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1470 \cs_set_eq:NN \Block \@@_Block:
1471 \cs_set_eq:NN \rotate \@@_rotate:
1472 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1473 \cs_set_eq:NN \dotfill \@@_dotfill:
1474 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1475 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1476 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1477 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1478 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1479 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1480 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1481 \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.

```

1482 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1483 \hook_gput_code:nnn { env / tabular / begin } { . }
1484 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }

```

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 (that remember that the caption will be composed *after* the array!).

```

1485 \tl_if_exist:NT \l_@@_note_in_caption_tl
1486 {
1487   \tl_if_empty:NF \l_@@_note_in_caption_tl
1488   {
1489     \int_gset_eq:NN \g_@@_notes_caption_int
1490     { \l_@@_note_in_caption_tl }
1491     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1492   }
1493 }

```

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).

```

1494 \seq_gclear:N \g_@@_multicolumn_cells_seq
1495 \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).

```

1496 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

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

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

```

1497 \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.

```

1498 \int_gzero_new:N \g_@@_col_total_int
1499 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1500 \@@_renew_NC@rewrite@S:
1501 \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.

```

1502   \tl_gclear_new:N \g_@@_Cdots_lines_tl
1503   \tl_gclear_new:N \g_@@_Ldots_lines_tl
1504   \tl_gclear_new:N \g_@@_Vdots_lines_tl
1505   \tl_gclear_new:N \g_@@_Ddots_lines_tl
1506   \tl_gclear_new:N \g_@@_Iddots_lines_tl
1507   \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1508   \tl_gclear:N \g_nicematrix_code_before_tl
1509   \tl_gclear:N \g_@@_pre_code_before_tl
1510 }

```

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

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

```

1511 \cs_new_protected:Npn \@@_pre_array:
1512 {
1513   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1514   \int_gzero_new:N \c@iRow
1515   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1516   \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).

```

1517   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1518   {
1519     \bool_set_true:N \l_@@_last_row_without_value_bool
1520     \bool_if:NT \g_@@_aux_found_bool
1521       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1522   }
1523   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1524   {
1525     \bool_if:NT \g_@@_aux_found_bool
1526       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1527   }

```

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”.

```

1528   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1529   {
1530     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1531     {
1532       \dim_gset:Nn \g_@@_ht_last_row_dim
1533       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1534       \dim_gset:Nn \g_@@_dp_last_row_dim
1535       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1536     }
1537   }

1538   \seq_gclear:N \g_@@_cols_vlism_seq
1539   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1540   \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.

```
1541 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

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

```
1542 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
```

```
1543 \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.

```
1544 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value  $-2$  is important.

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

```
1545 \@@_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.

```
1546 \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]`).

```
1547 \dim_zero_new:N \l_@@_left_delim_dim
1548 \dim_zero_new:N \l_@@_right_delim_dim
1549 \bool_if:NTF \g_@@_NiceArray_bool
1550 {
1551   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1552   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1553 }
1554 {
```

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

```
1555 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1556 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1557 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1558 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1559 }
```

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).

```
1560 \hbox_set:Nw \l_@@_the_array_box
1561 \skip_horizontal:N \l_@@_left_margin_dim
1562 \skip_horizontal:N \l_@@_extra_left_margin_dim
1563 \c_math_toggle_token
1564 \bool_if:NTF \l_@@_light_syntax_bool
1565 { \use:c { @@-light-syntax } }
1566 { \use:c { @@-normal-syntax } }
1567 }
```

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

```
1568 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1569 {
1570   \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1571   \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...

```
1572 \@@_pre_array:
1573 }
```

## 9 The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed.

```
1574 \cs_new_protected:Npn \@@_pre_code_before:
1575 {
```

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).

```
1576 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1577 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1578 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1579 \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.

```
1580 \pgfsys@markposition { \@@_env: - position }
1581 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1582 \pgfpicture
1583 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1584 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1585 {
1586   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1587   \pgfcoordinate { \@@_env: - row - ##1 }
1588   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1589 }
```

Now, the recreation of the `col` nodes.

```
1590 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1591 {
1592   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1593   \pgfcoordinate { \@@_env: - col - ##1 }
1594   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1595 }
```

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

```
1596 \@@_create_diag_nodes:
```

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

```
1597 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1598 \endpgfpicture
```

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

```
1599 \@@_create_blocks_nodes:
```

```

1600 \IfPackageLoadedTF { tikz }
1601 {
1602     \tikzset
1603     {
1604         every-picture / .style =
1605         { overlay , name-prefix = \@@_env: - }
1606     }
1607 }
1608 { }
1609 \cs_set_eq:NN \cellcolor \@@_cellcolor
1610 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1611 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1612 \cs_set_eq:NN \rowcolor \@@_rowcolor
1613 \cs_set_eq:NN \rowcolors \@@_rowcolors
1614 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1615 \cs_set_eq:NN \arraycolor \@@_arraycolor
1616 \cs_set_eq:NN \columncolor \@@_columncolor
1617 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1618 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1619 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1620 }

```

```

1621 \cs_new_protected:Npn \@@_exec_code_before:
1622 {
1623     \seq_gclear_new:N \g_@@_colors_seq
1624     \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1625     \group_begin:

```

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

```

1626     \bool_if:NT \l_@@_NiceTabular_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`).

```

1627     \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1628     {
1629         \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1630         \@@_rescan_for_spanish:N \l_@@_code_before_tl
1631     }

```

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`.

```

1632     \exp_last_unbraced:NV \@@_CodeBefore_keys:
1633     \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.

```

1634     \@@_actually_color:
1635     \l_@@_code_before_tl
1636     \q_stop
1637     \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1638     \group_end:
1639     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1640     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1641 }

```

```

1642 \keys_define:nn { NiceMatrix / CodeBefore }
1643 {
1644   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1645   create-cell-nodes .default:n = true ,
1646   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1647   sub-matrix .value_required:n = true ,
1648   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1649   delimiters / color .value_required:n = true ,
1650   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1651 }
1652 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1653 {
1654   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1655   \@@_CodeBefore:w
1656 }

```

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 `\CodeAfter`, excepted, of course, if we are in the first compilation.

```

1657 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1658 {
1659   \bool_if:NT \g_@@_aux_found_bool
1660   {
1661     \@@_pre_code_before:
1662     #1
1663   }
1664 }

```

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.

```

1665 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1666 {
1667   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1668   {
1669     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1670     \pgfcoordinate { \@@_env: - row - ##1 - base }
1671     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1672     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1673     {
1674       \cs_if_exist:cT
1675       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1676       {
1677         \pgfsys@getposition
1678         { \@@_env: - ##1 - #####1 - NW }
1679         \@@_node_position:
1680         \pgfsys@getposition
1681         { \@@_env: - ##1 - #####1 - SE }
1682         \@@_node_position_i:
1683         \@@_pgf_rect_node:nnn
1684         { \@@_env: - ##1 - #####1 }
1685         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1686         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1687       }
1688     }
1689   }
1690   \int_step_inline:nn \c@iRow
1691   {
1692     \pgfnodealias
1693     { \@@_env: - ##1 - last }

```

```

1694         { \@@_env: - ##1 - \int_use:N \c@jCol }
1695     }
1696     \int_step_inline:nn \c@jCol
1697     {
1698         \pgfnodealias
1699         { \@@_env: - last - ##1 }
1700         { \@@_env: - \int_use:N \c@iRow - ##1 }
1701     }
1702     \@@_create_extra_nodes:
1703 }

```

```

1704 \cs_new_protected:Npn \@@_create_blocks_nodes:
1705 {
1706     \pgfpicture
1707     \pgf@relevantforpicturesizefalse
1708     \pgfrememberpicturepositiononpagetrue
1709     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1710     { \@@_create_one_block_node:nnnnn ##1 }
1711     \endpgfpicture
1712 }

```

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>

```

1713 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1714 {
1715     \tl_if_empty:nF { #5 }
1716     {
1717         \@@_qpoint:n { col - #2 }
1718         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1719         \@@_qpoint:n { #1 }
1720         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1721         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1722         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1723         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1724         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1725         \@@_pgf_rect_node:nnnnn
1726         { \@@_env: - #5 }
1727         { \dim_use:N \l_tmpa_dim }
1728         { \dim_use:N \l_tmpb_dim }
1729         { \dim_use:N \l_@@_tmpc_dim }
1730         { \dim_use:N \l_@@_tmpd_dim }
1731     }
1732 }

```

```

1733 \cs_new_protected:Npn \@@_patch_for_revtext:
1734 {
1735     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1736     \cs_set_eq:NN \insert@column \insert@column@array
1737     \cs_set_eq:NN \@classx \@classx@array
1738     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1739     \cs_set_eq:NN \@arraycr \@arraycr@array
1740     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1741     \cs_set_eq:NN \array \array@array
1742     \cs_set_eq:NN \@array \@array@array
1743     \cs_set_eq:NN \@tabular \@tabular@array
1744     \cs_set_eq:NN \@mkpream \@mkpream@array
1745     \cs_set_eq:NN \endarray \endarray@array
1746     \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }

```

---

<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).



```

1747 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1748 }

```

## 10 The environment `\NiceArrayWithDelims`

```

1749 \NewDocumentEnvironment { NiceArrayWithDelims }
1750 { m m O { } m ! O { } t \CodeBefore }
1751 {
1752 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1753 \@@_provide_pgfsyspdfmark:
1754 \bool_if:NT \c_@@_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.

```

1755 \bgroup

1756 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1757 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1758 \tl_gset:Nn \g_@@_preamble_tl { #4 }

1759 \int_gzero:N \g_@@_block_box_int
1760 \dim_zero:N \g_@@_width_last_col_dim
1761 \dim_zero:N \g_@@_width_first_col_dim
1762 \bool_gset_false:N \g_@@_row_of_col_done_bool
1763 \str_if_empty:NT \g_@@_name_env_str
1764 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1765 \bool_if:NTF \l_@@_NiceTabular_bool
1766 \mode_leave_vertical:
1767 \@@_test_if_math_mode:
1768 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1769 \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.

```

1770 \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.

```

1771 \cs_if_exist:NT \tikz@library@external@loaded
1772 {
1773 \tikzexternaldisable
1774 \cs_if_exist:NT \ifstandalone
1775 { \tikzset { external / optimize = false } }
1776 }

```

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

```

1777 \int_gincr:N \g_@@_env_int
1778 \bool_if:NF \l_@@_block_auto_columns_width_bool
1779 { \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`).

---

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

```

1780 \seq_gclear:N \g_@@_blocks_seq
1781 \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`.

```

1782 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1783 \seq_gclear:N \g_@@_pos_of_xdots_seq
1784 \tl_gclear_new:N \g_@@_code_before_tl
1785 \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.

```

1786 \bool_gset_false:N \g_@@_aux_found_bool
1787 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1788 {
1789     \bool_gset_true:N \g_@@_aux_found_bool
1790     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1791 }

```

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.

```

1792 \tl_gclear:N \g_@@_aux_tl
1793 \tl_if_empty:NF \g_@@_code_before_tl
1794 {
1795     \bool_set_true:N \l_@@_code_before_bool
1796     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1797 }
1798 \tl_if_empty:NF \g_@@_pre_code_before_tl
1799 { \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`.

```

1800 \bool_if:NTF \g_@@_NiceArray_bool
1801 { \keys_set:nn { NiceMatrix / NiceArray } }
1802 { \keys_set:nn { NiceMatrix / pNiceArray } }
1803 { #3 , #5 }

1804 \@@_set_CT@arc@:V \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:.`

```

1805 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1806 }

```

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

```

1807 {
1808     \bool_if:NTF \l_@@_light_syntax_bool
1809     { \use:c { end @@-light-syntax } }
1810     { \use:c { end @@-normal-syntax } }
1811     \c_math_toggle_token
1812     \skip_horizontal:N \l_@@_right_margin_dim
1813     \skip_horizontal:N \l_@@_extra_right_margin_dim
1814     \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.

```

1815 \bool_if:NT \l_@@_width_used_bool
1816 {

```

```

1817     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1818     { \@@_error_or_warning:n { width~without~X~columns } }
1819 }

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  $n$ .

1820     \int_compare:nNnT \g_@@_total_X_weight_int > 0
1821     {
1822         \tl_gput_right:Nx \g_@@_aux_tl
1823         {
1824             \bool_set_true:N \l_@@_X_columns_aux_bool
1825             \dim_set:Nn \l_@@_X_columns_dim
1826             {
1827                 \dim_compare:nNnTF
1828                 {
1829                     \dim_abs:n
1830                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1831                 }
1832                 <
1833                 { 0.001 pt }
1834                 { \dim_use:N \l_@@_X_columns_dim }
1835                 {
1836                     \dim_eval:n
1837                     {
1838                         ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1839                         / \int_use:N \g_@@_total_X_weight_int
1840                         + \l_@@_X_columns_dim
1841                     }
1842                 }
1843             }
1844         }
1845     }

```

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).

```

1846     \int_compare:nNnT \l_@@_last_row_int > { -2 }
1847     {
1848         \bool_if:NF \l_@@_last_row_without_value_bool
1849         {
1850             \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1851             {
1852                 \@@_error:n { Wrong~last~row }
1853                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1854             }
1855         }
1856     }

```

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>

```

1857     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1858     \bool_if:nTF \g_@@_last_col_found_bool
1859     { \int_gdecr:N \c@jCol }
1860     {
1861         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1862         { \@@_error:n { last~col~not~used } }
1863     }

```

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

```

1864     \int_gset_eq:NN \g_@@_row_total_int \c@iRow

```

---

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

```
1865 \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. 84).

```
1866 \int_compare:nNnT \l_@@_first_col_int = 0
1867 {
1868   \skip_horizontal:N \col@sep
1869   \skip_horizontal:N \g_@@_width_first_col_dim
1870 }
```

The construction of the real box is different when `\g_@@_NiceArray_bool` is true (`{NiceArray}` or `{NiceTabular}`) and in the other environments because, in `{NiceArray}` or `{NiceTabular}`, we have no delimiter to put (but we have tabular notes to put). We begin with this case.

```
1871 \bool_if:NTF \g_@@_NiceArray_bool
1872 {
1873   \str_case:VnF \l_@@_baseline_tl
1874   {
1875     b \@@_use_arraybox_with_notes_b:
1876     c \@@_use_arraybox_with_notes_c:
1877   }
1878   \@@_use_arraybox_with_notes:
1879 }
```

Now, in the case of an environment `{pNiceArray}`, `{bNiceArray}`, etc. We compute `\l_tmpa_dim` which is the total height of the “first row” above the array (when the key `first-row` is used).

```
1880 {
1881   \int_compare:nNnTF \l_@@_first_row_int = 0
1882   {
1883     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1884     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1885   }
1886   { \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>

```
1887 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1888 {
1889   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1890   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1891 }
1892 { \dim_zero:N \l_tmpb_dim }
1893 \hbox_set:Nn \l_tmpa_box
1894 {
1895   \c_math_toggle_token
1896   \@@_color:V \l_@@_delimiters_color_tl
1897   \exp_after:wN \left \g_@@_left_delim_tl
1898   \vcenter
1899   {
```

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.

```
1900   \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1901   \hbox
1902   {
1903     \bool_if:NTF \l_@@_NiceTabular_bool
1904     { \skip_horizontal:N -\tabcolsep }
1905     { \skip_horizontal:N -\arraycolsep }
1906     \@@_use_arraybox_with_notes_c:
1907     \bool_if:NTF \l_@@_NiceTabular_bool
1908     { \skip_horizontal:N -\tabcolsep }
```

---

<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).

```

1909         { \skip_horizontal:N -\arraycolsep }
1910     }

```

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

```

1911         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1912     }

```

Curiously, we have to put again the following specification of color. Otherwise, with XeLaTeX (and not with the other engines), the closing delimiter is not colored.

```

1913         \@@_color:V \l_@@_delimiters_color_tl
1914         \exp_after:wN \right \g_@@_right_delim_tl
1915         \c_math_toggle_token
1916     }

```

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.

```

1917     \bool_if:NTF \l_@@_delimiters_max_width_bool
1918     {
1919         \@@_put_box_in_flow_bis:nn
1920         \g_@@_left_delim_tl \g_@@_right_delim_tl
1921     }
1922     \@@_put_box_in_flow:
1923 }

```

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. 85).

```

1924     \bool_if:NT \g_@@_last_col_found_bool
1925     {
1926         \skip_horizontal:N \g_@@_width_last_col_dim
1927         \skip_horizontal:N \col@sep
1928     }
1929     \bool_if:NF \l_@@_Matrix_bool
1930     {
1931         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1932         { \@@_warning_gredirect_none:n { columns-not-used } }
1933     }
1934     \@@_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.

```

1935     \egroup

```

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

```

1936     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1937     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1938     \iow_now:Nx \@mainaux
1939     {
1940         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1941         { \exp_not:V \g_@@_aux_tl }
1942     }
1943     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1944     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1945 }

```

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

## 11 We construct the preamble of the array

The transformation of the preamble is an operation in several steps.<sup>10</sup>

The preamble given by the final user is in `\g_@@_preamble_tl` and the modified version will be stored in `\g_@@_preamble_tl` also.

```
1946 \cs_new_protected:Npn \@@_transform_preamble:
1947 {
```

First, we will do an “expansion” of the preamble with the tools of the package `array` itself. This “expansion” will expand all the constructions with `*` and all column types (defined by the user or by various packages using `\newcolumntype`).

Since we use the tools of `array` to do this expansion, we will have a programming which is not in the style of the L3 programming layer.

We redefine the column types `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able to use the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```
1948 \group_begin:
```

If we are in an environment without explicit preamble, we have nothing to do (excepted the treatment on both sides of the preamble which will be done at the end).

```
1949 \bool_if:NF \l_@@_Matrix_bool
1950 {
1951   \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1952   \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
```

If the package `varwidth` has defined the column type `V`, we protect from expansion by redefining it to `\@@_V:` (which will be caught by our system).

```
1953 \cs_if_exist:NT \NC@find@V { \@@_newcolumntype V { \@@_V: } }
```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by the L3 programming layer).

```
1954 \exp_args:NV \@temptokena \g_@@_preamble_tl
```

Initialisation of a flag used by `array` to detect the end of the expansion.

```
1955 \@tempswatrue
```

The following line actually does the expansion (it’s has been copied from `array.sty`). The expanded version is still in `\@temptokena`.

```
1956 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
```

Now, we have to “patch” that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker `\q_stop` and we will consume these tokens constructing the (new form of the) preamble in `\g_@@_preamble_tl`. This is done recursively with the command `\@@_patch_preamble:n`. In the same time, we will count the columns with the counter `\c@jCol`.

```
1957 \int_gzero:N \c@jCol
1958 \tl_gclear:N \g_@@_preamble_tl
```

---

<sup>10</sup>Be careful: the transformation of the preamble may also have by-side effects, for example, the boolean `\g_@@_NiceArray_bool` will be set to `false` if we detect in the preamble a delimiter at the beginning or at the end.

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

```

1959     \bool_gset_false:N \g_tmpb_bool
1960     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1961     {
1962         \tl_gset:Nn \g_@@_preamble_tl
1963         { ! { \skip_horizontal:N \arrayrulewidth } }
1964     }
1965     {
1966         \clist_if_in:NnT \l_@@_vlines_clist 1
1967         {
1968             \tl_gset:Nn \g_@@_preamble_tl
1969             { ! { \skip_horizontal:N \arrayrulewidth } }
1970         }
1971     }

```

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

```

1972     \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1973     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

1974     \int_zero:N \l_tmpa_int

```

Now, we actually patch the preamble (and it is constructed in `\g_@@_preamble_tl`).

```

1975     \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1976     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1977 }

```

Now, we replace `\columncolor` by `\@@_columncolor_preamble`.

```

1978     \bool_if:NT \l_@@_colortbl_like_bool
1979     {
1980         \regex_replace_all:NnN
1981         \c_@@_columncolor_regex
1982         { \c { @@_columncolor_preamble } }
1983         \g_@@_preamble_tl
1984     }

```

Now, we can close the TeX group which was opened for the redefinition of the columns of type `w` and `W`.

```

1985     \group_end:

```

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

```

1986     \bool_lazy_or:nnT
1987     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1988     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1989     { \bool_gset_false:N \g_@@_NiceArray_bool }

```

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

```

1990     \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).

```

1991     \int_compare:nNnTF \l_@@_first_col_int = 0
1992     { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1993     {
1994         \bool_lazy_all:nT
1995         {
1996             \g_@@_NiceArray_bool
1997             { \bool_not_p:n \l_@@_NiceTabular_bool }
1998             { \tl_if_empty_p:N \l_@@_vlines_clist }

```

```

1999         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2000     }
2001     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
2002 }
2003 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2004 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
2005 {
2006     \bool_lazy_all:nT
2007     {
2008         \g_@@_NiceArray_bool
2009         { \bool_not_p:n \l_@@_NiceTabular_bool }
2010         { \tl_if_empty_p:N \l_@@_vlines_clist }
2011         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2012     }
2013     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
2014 }

```

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`).

```

2015     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2016     {
2017         \tl_gput_right:Nn \g_@@_preamble_tl
2018         { > { \@@_error_too_much_cols: } 1 }
2019     }
2020 }

```

The command `\@@_patch_preamble:n` is the main function for the transformation of the preamble. It is recursive.

```

2021 \cs_new_protected:Npn \@@_patch_preamble:n #1
2022 {
2023     \str_case:nnF { #1 }
2024     {
2025         c      { \@@_patch_preamble_i:n #1 }
2026         l      { \@@_patch_preamble_i:n #1 }
2027         r      { \@@_patch_preamble_i:n #1 }
2028         >      { \@@_patch_preamble_xiv:n }
2029         !      { \@@_patch_preamble_ii:nn #1 }
2030         @      { \@@_patch_preamble_ii:nn #1 }
2031         |      { \@@_patch_preamble_iii:n #1 }
2032         p      { \@@_patch_preamble_iv:n #1 }
2033         b      { \@@_patch_preamble_iv:n #1 }
2034         m      { \@@_patch_preamble_iv:n #1 }
2035         \@@_V: { \@@_patch_preamble_v:n }
2036         V      { \@@_patch_preamble_v:n }
2037         \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
2038         \@@_W: { \@@_patch_preamble_vi:nnnn { \@@_special_W: } #1 }
2039         \@@_S: { \@@_patch_preamble_vii:n }
2040         (      { \@@_patch_preamble_viii:nn #1 }
2041         [      { \@@_patch_preamble_viii:nn #1 }
2042         \{      { \@@_patch_preamble_viii:nn #1 }
2043         \left  { \@@_patch_preamble_viii:nn }
2044         )      { \@@_patch_preamble_ix:nn #1 }
2045         ]      { \@@_patch_preamble_ix:nn #1 }
2046         \}      { \@@_patch_preamble_ix:nn #1 }
2047         \right { \@@_patch_preamble_ix:nn }
2048         X      { \@@_patch_preamble_x:n }

```

When `tabularx` is loaded, a local redefinition of the specifier `X` is done to replace `X` by `\@@_X`. Thus, our column type `X` will be used in the `{NiceTabularX}`.

```

2049     \@@_X      { \@@_patch_preamble_x:n }
2050     \q_stop { }
2051 }

```



```

2052 {
2053   \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2054   {
2055     \seq_gput_right:Nx \g_@@_cols_vlism_seq
2056     { \int_eval:n { \c@jCol + 1 } }
2057     \tl_gput_right:Nx \g_@@_preamble_tl
2058     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2059     \@@_patch_preamble:n
2060   }

```

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs. Among the keys available in that list, there is the key `letter`. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix/ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

2061 {
2062   \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2063   {
2064     \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2065     \@@_patch_preamble:n
2066   }
2067   {
2068     \tl_if_eq:nnT { #1 } { S }
2069     { \@@_fatal:n { unknown~column~type~S } }
2070     { \@@_fatal:nn { unknown~column~type } { #1 } }
2071   }
2072 }
2073 }
2074 }

```

Now, we will list all the auxiliary functions for the different types of entries in the preamble of the array.

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

```

2075 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2076 {
2077   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2078   \tl_gclear:N \g_@@_pre_cell_tl
2079   \tl_gput_right:Nn \g_@@_preamble_tl
2080   {
2081     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2082     #1
2083     < \@@_cell_end:
2084   }

```

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

```

2085   \int_gincr:N \c@jCol
2086   \@@_patch_preamble_xi:n
2087 }

```

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

```

2088 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2089 {
2090   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2091   \@@_patch_preamble:n
2092 }

```

For `|`

```

2093 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2094 {

```

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

```

2095 \int_incr:N \l_tmpa_int
2096 \@@_patch_preamble_iii_i:n
2097 }
2098 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2099 {
2100 \str_if_eq:nnTF { #1 } |
2101 { \@@_patch_preamble_iii_i:n | }
2102 {
2103 \dim_set:Nn \l_tmpa_dim
2104 {
2105 \arrayrulewidth * \l_tmpa_int
2106 + \doublerulesep * ( \l_tmpa_int - 1)
2107 }
2108 \tl_gput_right:Nx \g_@@_preamble_tl
2109 {

```

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

```

2110 \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_tmpa_dim } } }
2111 }
2112 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2113 {
2114 \@@_vline:n
2115 {
2116 position = \int_eval:n { \c@jCol + 1 } ,
2117 multiplicity = \int_use:N \l_tmpa_int ,
2118 total-width = \dim_use:N \l_tmpa_dim % added 2022-08-06
2119 }

```

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.

```

2120 }
2121 \int_zero:N \l_tmpa_int
2122 \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2123 \@@_patch_preamble:n #1
2124 }
2125 }
2126 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2127 {
2128 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2129 \@@_patch_preamble:n
2130 }
2131 \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.

```

2132 \keys_define:nn { WithArrows / p-column }
2133 {
2134 r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2135 r .value_forbidden:n = true ,
2136 c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2137 c .value_forbidden:n = true ,
2138 l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2139 l .value_forbidden:n = true ,
2140 R .code:n =
2141 \IfPackageLoadedTF { ragged2e }
2142 { \str_set:Nn \l_@@_hpos_col_str { R } }
2143 {
2144 \@@_error_or_warning:n { ragged2e-not-loaded }
2145 \str_set:Nn \l_@@_hpos_col_str { r }
2146 } ,

```

```

2147 R .value_forbidden:n = true ,
2148 L .code:n =
2149   \IfPackageLoadedTF { ragged2e }
2150   { \str_set:Nn \l_@@_hpos_col_str { L } }
2151   {
2152     \@@_error_or_warning:n { ragged2e-not-loaded }
2153     \str_set:Nn \l_@@_hpos_col_str { l }
2154   } ,
2155 L .value_forbidden:n = true ,
2156 C .code:n =
2157   \IfPackageLoadedTF { ragged2e }
2158   { \str_set:Nn \l_@@_hpos_col_str { C } }
2159   {
2160     \@@_error_or_warning:n { ragged2e-not-loaded }
2161     \str_set:Nn \l_@@_hpos_col_str { c }
2162   } ,
2163 C .value_forbidden:n = true ,
2164 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2165 S .value_forbidden:n = true ,
2166 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2167 p .value_forbidden:n = true ,
2168 t .meta:n = p ,
2169 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2170 m .value_forbidden:n = true ,
2171 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2172 b .value_forbidden:n = true ,
2173 }

```

For p, b and m. The argument #1 is that value : p, b or m.

```

2174 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2175 {
2176   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2177   \@@_patch_preamble_iv_i:n
2178 }

2179 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2180 {
2181   \str_if_eq:nnTF { #1 } { [ ]
2182     { \@@_patch_preamble_iv_ii:w [ ]
2183       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2184     }
2185   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2186     { \@@_patch_preamble_iv_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.

```

2187 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2188 {

```

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).

```

2189   \str_set:Nn \l_@@_hpos_col_str { j }
2190   \tl_set:Nn \l_tmpa_tl { #1 }
2191   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2192   \@@_keys_p_column:V \l_tmpa_tl
2193   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2194 }

2195 \cs_new_protected:Npn \@@_keys_p_column:n #1
2196 { \keys_set_known:nn { WithArrows / p-column } { #1 } \l_tmpa_tl }
2197 \cs_generate_variant:Nn \@@_keys_p_column:n { V }

```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`.

```

2198 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2199 {
2200   \use:x
2201   {
2202     \@@_patch_preamble_iv_v:nnnnnnnn
2203     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2204     { \dim_eval:n { #1 } }
2205     {

```

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_str` which will provide the horizontal alignment of the column to which belongs the cell.

```

2206       \str_if_eq:VnTF \l_@@_hpos_col_str j
2207       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
2208       {
2209         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2210         { \str_lowercase:V \l_@@_hpos_col_str }
2211       }
2212       \str_case:Vn \l_@@_hpos_col_str
2213       {
2214         c { \exp_not:N \centering }
2215         l { \exp_not:N \raggedright }
2216         r { \exp_not:N \raggedleft }
2217         C { \exp_not:N \Centering }
2218         L { \exp_not:N \RaggedRight }
2219         R { \exp_not:N \RaggedLeft }
2220       }
2221     }
2222     { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2223     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2224     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2225     { #2 }
2226     {
2227       \str_case:VnF \l_@@_hpos_col_str
2228       {
2229         { j } { c }
2230         { si } { c }
2231       }

```

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

```

2232       { \str_lowercase:V \l_@@_hpos_col_str }
2233     }
2234   }

```

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

```

2235   \int_gincr:N \c@jCol
2236   \@@_patch_preamble_xi:n
2237 }

```

**#1** is the optional argument of `{minipage}` (or `{varwidth}`): `t` of `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_str` 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*.

```

2238 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2239 {
2240   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2241     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2242     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2243   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2244   \tl_gclear:N \g_@@_pre_cell_tl
2245   \tl_gput_right:Nn \g_@@_preamble_tl
2246     {
2247       > {

```

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.

```

2248       \dim_set:Nn \l_@@_col_width_dim { #2 }
2249       \@@_cell_begin:w
2250       \begin { #7 } [ #1 ] { #2 }

```

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

```

2251       \everypar
2252       {
2253         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2254         \everypar { }
2255       }

```

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

```

2256       #3

```

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

```

2257       \g_@@_row_style_tl
2258       \arraybackslash
2259       #5
2260     }
2261     #8
2262     < {
2263       #6

```

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

```

2264       \@finalstrut \@arstrutbox
2265       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
2266       \end { #7 }

```

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

```

2267       #4
2268       \@@_cell_end:
2269     }
2270   }
2271 }

```

```

2272 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2273 {
2274   \peek_meaning:NT \unskip
2275   {
2276     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2277       {
2278         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim

```

We put the following code in order to have a column with the correct width even when all the cells of the column are empty.

```

2279       \skip_horizontal:N \l_@@_col_width_dim
2280     }
2281   }
2282   #1
2283 }

```

```

2284 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2285 {
2286   \peek_meaning:NT \_siunitx_table_skip:n
2287   {
2288     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2289     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2290   }
2291   #1
2292 }

```

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.

```

2293 \cs_new_protected:Npn \@@_center_cell_box:
2294 {

```

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

```

2295   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2296   {
2297     \int_compare:nNnT
2298     { \box_ht:N \l_@@_cell_box }
2299     >

```

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).

```

2300     { \box_ht:N \strutbox }
2301     {
2302       \hbox_set:Nn \l_@@_cell_box
2303       {
2304         \box_move_down:nn
2305         {
2306           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2307             + \baselineskip ) / 2
2308         }
2309         { \box_use:N \l_@@_cell_box }
2310       }
2311     }
2312   }
2313 }

```

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

```

2314 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2315 {
2316   \str_if_eq:nnTF { #1 } { [ ] }
2317   { \@@_patch_preamble_v_i:w [ ] }
2318   { \@@_patch_preamble_v_i:w [ ] { #1 } }
2319 }
2320 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2321 { \@@_patch_preamble_v_ii:nn { #1 } }
2322 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2323 {
2324   \str_set:Nn \l_@@_vpos_col_str { p }
2325   \str_set:Nn \l_@@_hpos_col_str { j }
2326   \tl_set:Nn \l_tmpa_tl { #1 }
2327   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2328   \@@_keys_p_column:V \l_tmpa_tl
2329   \IfPackageLoadedTF { varwidth }
2330   { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2331   {

```

```

2332     \@@_error_or_warning:n { varwidth~not~loaded }
2333     \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2334   }
2335 }

```

For `w` and `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.

```

2336 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2337 {
2338   \str_if_eq:nnTF { #3 } { s }
2339     { \@@_patch_preamble_vi_i:nnnn { #1 } { #4 } }
2340     { \@@_patch_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2341 }

```

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.

```

2342 \cs_new_protected:Npn \@@_patch_preamble_vi_i:nnnn #1 #2
2343 {
2344   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2345   \tl_gclear:N \g_@@_pre_cell_tl
2346   \tl_gput_right:Nn \g_@@_preamble_tl
2347     {
2348       > {
2349         \dim_set:Nn \l_@@_col_width_dim { #2 }
2350         \@@_cell_begin:w
2351         \str_set:Nn \l_@@_hpos_cell_str { c }
2352       }
2353       c
2354       < {
2355         \@@_cell_end_for_w_s:
2356         #1
2357         \@@_adjust_size_box:
2358         \box_use_drop:N \l_@@_cell_box
2359       }
2360     }
2361   \int_gincr:N \c@jCol
2362   \@@_patch_preamble_xi:n
2363 }

```

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

```

2364 \cs_new_protected:Npn \@@_patch_preamble_vi_ii:nnnn #1 #2 #3 #4
2365 {
2366   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2367   \tl_gclear:N \g_@@_pre_cell_tl
2368   \tl_gput_right:Nn \g_@@_preamble_tl
2369     {
2370       > {

```

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.

```

2371         \dim_set:Nn \l_@@_col_width_dim { #4 }
2372         \hbox_set:Nw \l_@@_cell_box
2373         \@@_cell_begin:w
2374         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2375       }
2376       c
2377       < {
2378         \@@_cell_end:

```

```

2379         \hbox_set_end:
2380         % The following line is probably pointless
2381         % \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2382         #1
2383         \@@_adjust_size_box:
2384         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2385     }
2386 }

```

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

```

2387     \int_gincr:N \c@jCol
2388     \@@_patch_preamble_xi:n
2389 }

2390 \cs_new_protected:Npn \@@_special_W:
2391 {
2392     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2393     { \@@_warning:n { W-warning } }
2394 }

```

For \@@\_S:. If the user has used S[...], S has been replaced by \@@\_S: during the first expansion of the preamble (done with the tools of standard LaTeX and array).

```

2395 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2396 {
2397     \str_if_eq:nnTF { #1 } { [ ]
2398     { \@@_patch_preamble_vii_i:w [ ]
2399     { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2400     }

2401     \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2402     { \@@_patch_preamble_vii_ii:n { #1 } }

2403     \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2404     {
2405         \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2406         {
2407             \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2408             \tl_gclear:N \g_@@_pre_cell_tl
2409             \tl_gput_right:Nn \g_@@_preamble_tl
2410             {
2411                 > {
2412                     \@@_cell_begin:w
2413                     \keys_set:nn { siunitx } { #1 }
2414                     \siunitx_cell_begin:w
2415                 }
2416                 c
2417                 < { \siunitx_cell_end: \@@_cell_end: }
2418             }

```

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

```

2419     \int_gincr:N \c@jCol
2420     \@@_patch_preamble_xi:n
2421 }
2422 { \@@_fatal:n { Version-of-siunitx-too-old } }
2423 }

```

For (, [ and \{.

```

2424 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2425 {
2426     \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.

```

2427     \int_compare:nNnTF \c@jCol = \c_zero_int
2428     {
2429         \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2430         {

```



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

```

2431         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2432         \tl_gset:Nn \g_@@_right_delim_tl { . }
2433         \@@_patch_preamble:n #2
2434     }
2435     {
2436         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2437         \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2438     }
2439 }
2440 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2441 }
2442 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2443 {
2444     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2445     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2446     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2447     {
2448         \@@_error:nn { delimiter~after~opening } { #2 }
2449         \@@_patch_preamble:n
2450     }
2451     { \@@_patch_preamble:n #2 }
2452 }

```

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}`).

```

2453 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2454 {
2455     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2456     \tl_if_in:nnTF { ) ] \} } { #2 }
2457     { \@@_patch_preamble_ix_i:nnn #1 #2 }
2458     {
2459         \tl_if_eq:nnTF { \q_stop } { #2 }
2460         {
2461             \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2462             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2463             {
2464                 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2465                 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2466                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2467                 \@@_patch_preamble:n #2
2468             }
2469         }
2470         {
2471             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2472             { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2473             \tl_gput_right:Nx \g_@@_pre_code_after_tl
2474             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2475             \@@_patch_preamble:n #2
2476         }
2477     }
2478 }
2479 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2480 {
2481     \tl_if_eq:nnTF { \q_stop } { #3 }
2482     {
2483         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2484         {

```

```

2485 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2486 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2487 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2488 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2489 }
2490 {
2491 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2492 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2493 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2494 \@@_error:nn { double~closing~delimiter } { #2 }
2495 }
2496 }
2497 {
2498 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2499 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2500 \@@_error:nn { double~closing~delimiter } { #2 }
2501 \@@_patch_preamble:n #3
2502 }
2503 }

```

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.

```

2504 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2505 {
2506 \str_if_eq:nnTF { #1 } { [ ]
2507 { \@@_patch_preamble_x_i:w [ ]
2508 { \@@_patch_preamble_x_i:w [ ] #1 }
2509 }
2510 \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2511 { \@@_patch_preamble_x_ii: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).

```

2512 \keys_define:nn { WithArrows / X-column }
2513 { 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.

```

2514 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2515 {

```

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).

```

2516 \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).

```

2517 \tl_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.

```

2518 \int_zero_new:N \l_@@_weight_int
2519 \int_set:Nn \l_@@_weight_int { 1 }
2520 \tl_set:Nn \l_tmpa_tl { #1 }
2521 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2522 \@@_keys_p_column:V \l_tmpa_tl
2523 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2524 \int_compare:nNnT \l_@@_weight_int < 0
2525 {

```

```

2526 \@@_error_or_warning:n { negative-weight }
2527 \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2528 }
2529 \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).

```

2530 \bool_if:NTF \l_@@_X_columns_aux_bool
2531 {
2532   \exp_args:Nnx
2533   \@@_patch_preamble_iv_iv:nn
2534   { \l_@@_weight_int \l_@@_X_columns_dim }
2535   { minipage }
2536 }
2537 {
2538   \tl_gput_right:Nn \g_@@_preamble_tl
2539   {
2540     > {
2541       \@@_cell_begin:w
2542       \bool_set_true:N \l_@@_X_column_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`.

```

2543 \NotEmpty

```

The following code will nullify the box of the cell.

```

2544 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2545 { \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`.

```

2546 \begin { minipage } { 5 cm } \arraybackslash
2547 }
2548 c
2549 < {
2550   \end { minipage }
2551   \@@_cell_end:
2552 }
2553 }
2554 \int_gincr:N \c@jCol
2555 \@@_patch_preamble_xi:n
2556 }
2557 }

```

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 `@{...}`.

```

2558 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2559 {
2560   \str_if_eq:nnTF { #1 } { < }
2561   \@@_patch_preamble_xiii:n
2562   {
2563     \str_if_eq:nnTF { #1 } { @ }
2564     \@@_patch_preamble_xv:n
2565     {
2566       \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2567       {
2568         \tl_gput_right:Nn \g_@@_preamble_tl
2569         { ! { \skip_horizontal:N \arrayrulewidth } }
2570       }
2571       {
2572         \exp_args:NNx

```

```

2573         \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2574         {
2575             \tl_gput_right:Nn \g_@@_preamble_tl
2576             { ! { \skip_horizontal:N \arrayrulewidth } }
2577         }
2578     }
2579     \@@_patch_preamble:n { #1 }
2580 }
2581 }
2582 }
2583 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2584 {
2585     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2586     \@@_patch_preamble_xi:n
2587 }

```

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.

```

2588 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2589 {
2590     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2591     {
2592         \tl_gput_right:Nn \g_@@_preamble_tl
2593         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2594     }
2595     {
2596         \exp_args:NNx
2597         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2598         {
2599             \tl_gput_right:Nn \g_@@_preamble_tl
2600             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2601         }
2602         { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2603     }
2604     \@@_patch_preamble:n
2605 }
2606 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2607 {
2608     \group_begin:
2609     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2610     \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2611     \temptokena { #2 }
2612     \@tempswatrue
2613     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2614     \tl_gclear:N \g_@@_preamble_tl
2615     \exp_after:wN \@@_patch_m_preamble:n \the \temptokena \q_stop
2616     \group_end:
2617     \tl_set_eq:NN #1 \g_@@_preamble_tl
2618 }

```

## 12 The redefinition of `\multicolumn`

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

```

2619 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2620 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2621 \multispan { #1 }
2622 \begingroup
2623 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
2624 \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2625 \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

You do the expansion of the (small) preamble with the tools of `array`.

```

2626 \@temptokena = { #2 }
2627 \@tempswatrue
2628 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

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

```

2629 \tl_gclear:N \g_@@_preamble_tl
2630 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

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

```

2631 \exp_args:NV \@mkpream \g_@@_preamble_tl
2632 \@addtopreamble \@empty
2633 \endgroup

```

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

```

2634 \int_compare:nNnT { #1 } > 1
2635 {
2636   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2637   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2638   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2639   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2640   {
2641     {
2642       \int_compare:nNnTF \c@jCol = 0
2643       { \int_eval:n { \c@iRow + 1 } }
2644       { \int_use:N \c@iRow }
2645     }
2646     { \int_eval:n { \c@jCol + 1 } }
2647     {
2648       \int_compare:nNnTF \c@jCol = 0
2649       { \int_eval:n { \c@iRow + 1 } }
2650       { \int_use:N \c@iRow }
2651     }
2652     { \int_eval:n { \c@jCol + #1 } }
2653     { } % for the name of the block
2654   }
2655 }

```

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

```

2656 \cs_set:Npn \@sharp { #3 }
2657 \@arstrut
2658 \@preamble
2659 \null

```

We add some lines.

```

2660 \int_gadd:Nn \c@jCol { #1 - 1 }
2661 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2662 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2663 \ignorespaces
2664 }

```

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`.

```

2665 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2666 {
2667   \str_case:nnF { #1 }
2668   {
2669     c { \@@_patch_m_preamble_i:n #1 }
2670     l { \@@_patch_m_preamble_i:n #1 }
2671     r { \@@_patch_m_preamble_i:n #1 }
2672     > { \@@_patch_m_preamble_ii:nn #1 }
2673     ! { \@@_patch_m_preamble_ii:nn #1 }
2674     @ { \@@_patch_m_preamble_ii:nn #1 }
2675     | { \@@_patch_m_preamble_iii:n #1 }
2676     p { \@@_patch_m_preamble_iv:nnn t #1 }
2677     m { \@@_patch_m_preamble_iv:nnn c #1 }
2678     b { \@@_patch_m_preamble_iv:nnn b #1 }
2679     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2680     \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2681     \q_stop { }
2682   }
2683   {
2684     \tl_if_eq:nnT { #1 } { S }
2685     { \@@_fatal:n { unknown~column~type~S } }
2686     { \@@_fatal:nn { unknown~column~type } { #1 } }
2687   }
2688 }

```

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

```

2689 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2690 {
2691   \tl_gput_right:Nn \g_@@_preamble_tl
2692   {
2693     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2694     #1
2695     < \@@_cell_end:
2696   }

```

We test for the presence of a `<`.

```

2697   \@@_patch_m_preamble_x:n
2698 }

```

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

```

2699 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2700 {
2701   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2702   \@@_patch_m_preamble:n
2703 }

```

For `|`

```

2704 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2705 {
2706   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2707   \@@_patch_m_preamble:n
2708 }

```

For `p`, `m` and `b`

```

2709 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2710 {
2711   \tl_gput_right:Nn \g_@@_preamble_tl
2712   {
2713     > {
2714       \@@_cell_begin:w
2715       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }

```

```

2716         \mode_leave_vertical:
2717         \arraybackslash
2718         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2719     }
2720     c
2721     < {
2722         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2723         \end { minipage }
2724         \@@_cell_end:
2725     }
2726 }

```

We test for the presence of a <.

```

2727     \@@_patch_m_preamble_x:n
2728 }

```

For w and W

```

2729 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2730 {
2731     \tl_gput_right:Nn \g_@@_preamble_tl
2732     {
2733         > {
2734             \dim_set:Nn \l_@@_col_width_dim { #4 }
2735             \hbox_set:Nw \l_@@_cell_box
2736             \@@_cell_begin:w
2737             \str_set:Nn \l_@@_hpos_cell_str { #3 }
2738         }
2739         c
2740         < {
2741             \@@_cell_end:
2742             \hbox_set_end:
2743             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2744             #1
2745             \@@_adjust_size_box:
2746             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2747         }
2748     }

```

We test for the presence of a <.

```

2749     \@@_patch_m_preamble_x:n
2750 }

```

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

```

2751 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2752 {
2753     \str_if_eq:nnTF { #1 } { < }
2754     \@@_patch_m_preamble_ix:n
2755     { \@@_patch_m_preamble:n { #1 } }
2756 }
2757 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2758 {
2759     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2760     \@@_patch_m_preamble_x:n
2761 }

```

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).

```

2762 \cs_new_protected:Npn \@@_put_box_in_flow:
2763 {
2764     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2765     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }

```

```

2766 \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2767 { \box_use_drop:N \l_tmpa_box }
2768 \@@_put_box_in_flow_i:
2769 }

```

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).

```

2770 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2771 {
2772   \pgfpicture
2773   \@@_qpoint:n { row - 1 }
2774   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2775   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2776   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2777   \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).

```

2778   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2779   {
2780     \int_set:Nn \l_tmpa_int
2781     {
2782       \str_range:Nnn
2783       \l_@@_baseline_tl
2784       6
2785       { \tl_count:V \l_@@_baseline_tl }
2786     }
2787     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2788   }
2789   {
2790     \str_case:VnF \l_@@_baseline_tl
2791     {
2792       { t } { \int_set:Nn \l_tmpa_int 1 }
2793       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2794     }
2795     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2796     \bool_lazy_or:nnT
2797     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2798     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2799     {
2800       \@@_error:n { bad~value~for~baseline }
2801       \int_set:Nn \l_tmpa_int 1
2802     }
2803     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2804     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2805   }
2806   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2807   \endpgfpicture
2808   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2809   \box_use_drop:N \l_tmpa_box
2810 }

```

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).

```

2811 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2812 {

```



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.

```

2813 \bool_lazy_and:nNT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2814 {
2815   \box_set_wd:Nn \l_@@_the_array_box
2816   { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2817 }

```

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

```

2818 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2819 \bool_if:NT \l_@@_caption_above_bool
2820 {
2821   \tl_if_empty:NF \l_@@_caption_tl
2822   {
2823     \bool_set_false:N \g_@@_caption_finished_bool
2824     \int_gzero:N \c@tabularnote
2825     \@@_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).

```

2826     \int_compare:nNnT \g_@@_notes_caption_int > 0
2827     {
2828       \tl_gput_right:Nx \g_@@_aux_tl
2829       {
2830         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2831         { \int_use:N \g_@@_notes_caption_int }
2832       }
2833       \int_gzero:N \g_@@_notes_caption_int
2834     }
2835   }
2836 }

```

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

```

2837 \hbox
2838 {
2839   \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.

```

2840   \@@_create_extra_nodes:
2841   \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2842 }

```

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 twice its tabular).

```

2843 \bool_lazy_any:nT
2844 {
2845   { ! \seq_if_empty_p:N \g_@@_notes_seq }
2846   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2847   { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2848 }
2849 \@@_insert_tabularnotes:
2850 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2851 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2852 \end { minipage }
2853 }

```

```

2854 \cs_new_protected:Npn \@@_insert_caption:
2855 {
2856   \tl_if_empty:NF \l_@@_caption_tl
2857   {
2858     \cs_if_exist:NTF \@capytype
2859     { \@@_insert_caption_i: }
2860     { \@@_error:n { caption~outside~float } }
2861   }
2862 }

```

```

2863 \cs_new_protected:Npn \@@_insert_caption_i:
2864 {
2865   \group_begin:

```

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

```

2866   \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.

```

2867   \IfPackageLoadedTF { floatrow }
2868   { \cs_set_eq:NN \@makecaption \FR@makecaption }
2869   { }
2870   \tl_if_empty:NTF \l_@@_short_caption_tl
2871   { \caption }
2872   { \caption [ \l_@@_short_caption_tl ] }
2873   { \l_@@_caption_tl }
2874   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2875   \group_end:
2876 }

```

```

2877 \cs_new_protected:Npn \@@_tabularnote_error:n #1
2878 {
2879   \@@_error_or_warning:n { tabularnote~below~the~tabular }
2880   \@@_gredirect_none:n { tabularnote~below~the~tabular }
2881 }

```

```

2882 \cs_new_protected:Npn \@@_insert_tabularnotes:
2883 {
2884   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2885   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2886   \skip_vertical:N 0.65ex

```

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

```

2887   \group_begin:
2888   \l_@@_notes_code_before_tl
2889   \tl_if_empty:NF \g_@@_tabularnote_tl
2890   {
2891     \g_@@_tabularnote_tl \par
2892     \tl_gclear:N \g_@@_tabularnote_tl
2893   }

```

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.

```

2894   \int_compare:nNnT \c@tabularnote > 0
2895   {
2896     \bool_if:NTF \l_@@_notes_para_bool
2897     {
2898       \begin { tabularnotes* }
2899       \seq_map_inline:Nn \g_@@_notes_seq
2900       { \@@_one_tabularnote:nn ##1 }
2901       \strut
2902       \end { tabularnotes* }

```

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

```

2903     \par
2904   }
2905   {
2906     \tabularnotes
2907       \seq_map_inline:Nn \g_@@_notes_seq
2908       { \@@_one_tabularnote:nn ##1 }
2909     \strut
2910   \endtabularnotes
2911   }
2912 }
2913 \unskip
2914 \group_end:
2915 \bool_if:NT \l_@@_notes_bottomrule_bool
2916 {
2917   \IfPackageLoadedTF { booktabs }
2918   {

```

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

```

2919     \skip_vertical:N \aboverulesep

```

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

```

2920     { \CT@arc@ \hrule height \heavyrulewidth }
2921   }
2922   { \@@_error_or_warning:n { bottomrule-without-booktabs } }
2923 }
2924 \l_@@_notes_code_after_tl
2925 \seq_gclear:N \g_@@_notes_seq
2926 \seq_gclear:N \g_@@_notes_in_caption_seq
2927 \int_gzero:N \c@tabularnote
2928 }

```

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 curriification.

```

2929 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
2930 {
2931   \tl_if_novalue:nTF { #1 }
2932   { \item }
2933   { \item [ \@@_notes_label_in_list:n { #1 } ] }
2934 }

```

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`.

```

2935 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2936 {
2937   \pgfpicture
2938     \@@_qpoint:n { row - 1 }
2939     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2940     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2941     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2942   \endpgfpicture
2943   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2944   \int_compare:nNnT \l_@@_first_row_int = 0
2945   {
2946     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2947     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2948   }
2949   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2950 }

```

Now, the general case.

```

2951 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2952 {

```

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

```

2953 \tl_if_eq:NnT \l_@@_baseline_tl { t }
2954 { \tl_set:Nn \l_@@_baseline_tl { 1 } }

```

Now, we convert the value of `\l_@@_baseline_tl` (which should represent an integer) to an integer stored in `\l_tmpa_int`.

```

2955 \pgfpicture
2956 \@@_qpoint:n { row - 1 }
2957 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2958 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2959 {
2960   \int_set:Nn \l_tmpa_int
2961   {
2962     \str_range:Nnn
2963     \l_@@_baseline_tl
2964     6
2965     { \tl_count:V \l_@@_baseline_tl }
2966   }
2967   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2968 }
2969 {
2970   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2971   \bool_lazy_or:nnT
2972   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2973   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2974   {
2975     \@@_error:n { bad-value-for-baseline }
2976     \int_set:Nn \l_tmpa_int 1
2977   }
2978   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2979 }
2980 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2981 \endpgfpicture
2982 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2983 \int_compare:nNnT \l_@@_first_row_int = 0
2984 {
2985   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2986   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2987 }
2988 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2989 }

```

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.

```

2990 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2991 {

```

We will compute the real width of both delimiters used.

```

2992 \dim_zero_new:N \l_@@_real_left_delim_dim
2993 \dim_zero_new:N \l_@@_real_right_delim_dim
2994 \hbox_set:Nn \l_tmpb_box
2995 {
2996   \c_math_toggle_token
2997   \left #1
2998   \vcenter
2999   {
3000     \vbox_to_ht:nn
3001     { \box_ht_plus_dp:N \l_tmpa_box }
3002     { }

```

```

3003     }
3004     \right .
3005     \c_math_toggle_token
3006   }
3007   \dim_set:Nn \l_@@_real_left_delim_dim
3008   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3009   \hbox_set:Nn \l_tmpb_box
3010   {
3011     \c_math_toggle_token
3012     \left .
3013     \vbox_to_ht:nn
3014     { \box_ht_plus_dp:N \l_tmpa_box }
3015     { }
3016     \right #2
3017     \c_math_toggle_token
3018   }
3019   \dim_set:Nn \l_@@_real_right_delim_dim
3020   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3021   \skip_horizontal:N \l_@@_left_delim_dim
3022   \skip_horizontal:N -\l_@@_real_left_delim_dim
3023   \@@_put_box_in_flow:
3024   \skip_horizontal:N \l_@@_right_delim_dim
3025   \skip_horizontal:N -\l_@@_real_right_delim_dim
3026 }

```

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).

```

3027 \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).

```

3028 {
3029   \peek_remove_spaces:n
3030   {
3031     \peek_meaning:NTF \end
3032     \@@_analyze_end:Nn
3033     {
3034       \@@_transform_preamble:

```

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

```

3035       \@@_array:V \g_@@_preamble_tl
3036     }
3037   }
3038 }
3039 {
3040   \@@_create_col_nodes:
3041   \endarray
3042 }

```

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

```

3043 \NewDocumentEnvironment { @@-light-syntax } { b }
3044 {

```

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.

```

3045 \tl_if_empty:nT { #1 } { @@_fatal:n { empty-environment } }
3046 \tl_map_inline:nn { #1 }
3047 {
3048   \str_if_eq:nnT { ##1 } { & }
3049   { @@_fatal:n { ampersand-in-light-syntax } }
3050   \str_if_eq:nnT { ##1 } { \ }
3051   { @@_fatal:n { double-backslash-in-light-syntax } }
3052 }

```

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`.

```

3053 @@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3054 }

```

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.

```

3055 {
3056   @@_create_col_nodes:
3057   \endarray
3058 }
3059 \cs_new_protected:Npn @@_light_syntax_i:w #1\CodeAfter #2\q_stop
3060 {
3061   \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).

```

3062 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3063 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3064 \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3065 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3066 \tl_if_empty:NF \l_tmpa_tl
3067 { \seq_put_right:NV \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.

```

3068 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3069 { \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` (that part of the implementation has been changed in the version 6.11 of nicematrix in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`).

```

3070 \tl_clear_new:N \l_@@_new_body_tl
3071 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3072 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3073 @@_line_with_light_syntax:V \l_tmpa_tl

```

Now, the other rows (with the same treatment, excepted that we have to insert `\\` between the rows).

```

3074 \seq_map_inline:Nn \l_@@_rows_seq
3075 {
3076   \tl_put_right:Nn \l_@@_new_body_tl { \\ }
3077   \@@_line_with_light_syntax:n { ##1 }
3078 }
3079 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3080 {
3081   \int_set:Nn \l_@@_last_col_int
3082     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3083 }

```

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.

```

3084 \@@_transform_preamble:

```

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

```

3085 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3086 }
3087 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3088 {
3089   \seq_clear_new:N \l_@@_cells_seq
3090   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3091   \int_set:Nn \l_@@_nb_cols_int
3092     {
3093       \int_max:nn
3094         \l_@@_nb_cols_int
3095         { \seq_count:N \l_@@_cells_seq }
3096     }
3097   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3098   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3099   \seq_map_inline:Nn \l_@@_cells_seq
3100     { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3101 }
3102 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { V }

```

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`.

```

3103 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3104 {
3105   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3106     { \@@_fatal:n { empty~environment } }

```

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

```

3107 \end { #2 }
3108 }

```

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).

```

3109 \cs_new:Npn \@@_create_col_nodes:
3110 {
3111   \crcr
3112   \int_compare:nNnT \l_@@_first_col_int = 0
3113     {
3114       \omit
3115       \hbox_overlap_left:n
3116     }

```

```

3117         \bool_if:NT \l_@@_code_before_bool
3118         { \pgfsys@markposition { \@@_env: - col - 0 } }
3119         \pgfpicture
3120         \pgfrememberpicturerepositiononpagetrue
3121         \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3122         \str_if_empty:NF \l_@@_name_str
3123         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3124         \endpgfpicture
3125         \skip_horizontal:N 2\col@sep
3126         \skip_horizontal:N \g_@@_width_first_col_dim
3127     }
3128     &
3129 }
3130 \omit

```

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

```

3131     \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`).

```

3132     \int_compare:nNnTF \l_@@_first_col_int = 0
3133     {
3134         \bool_if:NT \l_@@_code_before_bool
3135         {
3136             \hbox
3137             {
3138                 \skip_horizontal:N -0.5\arrayrulewidth
3139                 \pgfsys@markposition { \@@_env: - col - 1 }
3140                 \skip_horizontal:N 0.5\arrayrulewidth
3141             }
3142         }
3143         \pgfpicture
3144         \pgfrememberpicturerepositiononpagetrue
3145         \pgfcoordinate { \@@_env: - col - 1 }
3146         { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3147         \str_if_empty:NF \l_@@_name_str
3148         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3149         \endpgfpicture
3150     }
3151     {
3152         \bool_if:NT \l_@@_code_before_bool
3153         {
3154             \hbox
3155             {
3156                 \skip_horizontal:N 0.5\arrayrulewidth
3157                 \pgfsys@markposition { \@@_env: - col - 1 }
3158                 \skip_horizontal:N -0.5\arrayrulewidth
3159             }
3160         }
3161         \pgfpicture
3162         \pgfrememberpicturerepositiononpagetrue
3163         \pgfcoordinate { \@@_env: - col - 1 }
3164         { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3165         \str_if_empty:NF \l_@@_name_str
3166         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3167         \endpgfpicture
3168     }

```

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 this 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 it will just after be erased by a fixed value in the concerned cases.



```

3169 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3170 \bool_if:NF \l_@@_auto_columns_width_bool
3171 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3172 {
3173   \bool_lazy_and:nnTF
3174     \l_@@_auto_columns_width_bool
3175     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3176     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3177     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3178   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3179 }
3180 \skip_horizontal:N \g_tmpa_skip
3181 \hbox
3182 {
3183   \bool_if:NT \l_@@_code_before_bool
3184   {
3185     \hbox
3186     {
3187       \skip_horizontal:N -0.5\arrayrulewidth
3188       \pgfsys@markposition { \@@_env: - col - 2 }
3189       \skip_horizontal:N 0.5\arrayrulewidth
3190     }
3191   }
3192   \pgfpicture
3193   \pgfrememberpicturepositiononpagetrue
3194   \pgfcoordinate { \@@_env: - col - 2 }
3195   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3196   \str_if_empty:NF \l_@@_name_str
3197   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3198   \endpgfpicture
3199 }

```

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.

```

3200 \int_gset:Nn \g_tmpa_int 1
3201 \bool_if:NTF \g_@@_last_col_found_bool
3202 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3203 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3204 {
3205   &
3206   \omit
3207   \int_gincr:N \g_tmpa_int

```

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

```

3208 \skip_horizontal:N \g_tmpa_skip
3209 \bool_if:NT \l_@@_code_before_bool
3210 {
3211   \hbox
3212   {
3213     \skip_horizontal:N -0.5\arrayrulewidth
3214     \pgfsys@markposition
3215     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3216     \skip_horizontal:N 0.5\arrayrulewidth
3217   }
3218 }

```

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

```

3219 \pgfpicture
3220 \pgfrememberpicturepositiononpagetrue
3221 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3222 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3223 \str_if_empty:NF \l_@@_name_str
3224 {
3225   \pgfnodealias
3226   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }

```

```

3227         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3228     }
3229 \endpgfpicture
3230 }

3231 &
3232 \omit

```

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

```

3233 \int_compare:nNt \g_@@_col_total_int = 1
3234 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3235 \skip_horizontal:N \g_tmpa_skip
3236 \int_gincr:N \g_tmpa_int
3237 \bool_lazy_all:nT
3238 {
3239     \g_@@_NiceArray_bool
3240     { \bool_not_p:n \l_@@_NiceTabular_bool }
3241     { \clist_if_empty_p:N \l_@@_vlines_clist }
3242     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3243     { ! \l_@@_bar_at_end_of_pream_bool }
3244 }
3245 { \skip_horizontal:N -\col@sep }
3246 \bool_if:NT \l_@@_code_before_bool
3247 {
3248     \hbox
3249     {
3250         \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.

```

3251         \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3252         { \skip_horizontal:N -\arraycolsep }
3253     \pgfsys@markposition
3254     { \@@_env: - col - \int_eval:n {
3255         \g_tmpa_int + 1 } }
3256     \skip_horizontal:N 0.5\arrayrulewidth
3257     \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3258     { \skip_horizontal:N \arraycolsep }
3259 }
3260 }
3261 \pgfpicture
3262 \pgfrememberpicturepositiononpagetrue
3263 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3264 {
3265     \bool_lazy_and:nnTF \l_@@_Matrix_bool \g_@@_NiceArray_bool
3266     {
3267         \pgfpoint
3268         { - 0.5 \arrayrulewidth - \arraycolsep }
3269         \c_zero_dim
3270     }
3271     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3272 }
3273 \str_if_empty:NF \l_@@_name_str
3274 {
3275     \pgfnodelias
3276     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3277     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3278 }
3279 \endpgfpicture

```

```

3280 \bool_if:NT \g_@@_last_col_found_bool
3281 {
3282   \hbox_overlap_right:n
3283   {
3284     \skip_horizontal:N \g_@@_width_last_col_dim
3285     \bool_if:NT \l_@@_code_before_bool
3286     {
3287       \pgfsys@markposition
3288       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3289     }
3290     \pgfpicture
3291     \pgfrememberpicturepositiononpagetrue
3292     \pgfcoordinate
3293     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3294     \pgfpointorigin
3295     \str_if_empty:NF \l_@@_name_str
3296     {
3297       \pgfnodealias
3298       {
3299         \l_@@_name_str - col
3300         - \int_eval:n { \g_@@_col_total_int + 1 }
3301       }
3302       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3303     }
3304     \endpgfpicture
3305   }
3306 }
3307 \cr
3308 }

```

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

```

3309 \tl_const:Nn \c_@@_preamble_first_col_tl
3310 {
3311   >
3312   {

```

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).

```

3313 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3314 \bool_gset_true:N \g_@@_after_col_zero_bool
3315 \@@_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.

```

3316 \hbox_set:Nw \l_@@_cell_box
3317 \@@_math_toggle_token:
3318 \bool_if:NT \l_@@_small_bool \scriptstyle

```

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”.

```

3319 \bool_lazy_and:nnT
3320 { \int_compare_p:nNn \c@iRow > 0 }
3321 {
3322   \bool_lazy_or_p:nn
3323   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3324   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3325 }
3326 {
3327   \l_@@_code_for_first_col_tl
3328   \xglobal \colorlet { nicematrix-first-col } { . }
3329 }
3330 }

```

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`.

```

3331   l
3332   <
3333   {
3334     \@@_math_toggle_token:
3335     \hbox_set_end:
3336     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3337     \@@_adjust_size_box:
3338     \@@_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.

```

3339     \dim_gset:Nn \g_@@_width_first_col_dim
3340     { \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.

```

3341     \hbox_overlap_left:n
3342     {
3343       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3344       \@@_node_for_cell:
3345       { \box_use_drop:N \l_@@_cell_box }
3346       \skip_horizontal:N \l_@@_left_delim_dim
3347       \skip_horizontal:N \l_@@_left_margin_dim
3348       \skip_horizontal:N \l_@@_extra_left_margin_dim
3349     }
3350     \bool_gset_false:N \g_@@_empty_cell_bool
3351     \skip_horizontal:N -2\col@sep
3352   }
3353 }

```

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

```

3354 \tl_const:Nn \c_@@_preamble_last_col_tl
3355 {
3356   >
3357   {

```

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).

```

3358     \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.

```

3359     \bool_gset_true:N \g_@@_last_col_found_bool
3360     \int_gincr:N \c@jCol
3361     \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.

```

3362     \hbox_set:Nw \l_@@_cell_box
3363     \@@_math_toggle_token:
3364     \bool_if:NT \l_@@_small_bool \scriptstyle

```

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”.

```

3365     \int_compare:nNnT \c@iRow > 0
3366     {
3367       \bool_lazy_or:nnT
3368       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3369       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3370       {
3371         \l_@@_code_for_last_col_tl
3372         \xglobal \colorlet { nicematrix-last-col } { . }
3373       }
3374     }
3375   }
3376   l

```

```

3377 <
3378 {
3379   \@@_math_toggle_token:
3380   \hbox_set_end:
3381   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3382   \@@_adjust_size_box:
3383   \@@_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.

```

3384   \dim_gset:Nn \g_@@_width_last_col_dim
3385   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3386   \skip_horizontal:N -2\col@sep

```

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

```

3387   \hbox_overlap_right:n
3388   {
3389     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3390     {
3391       \skip_horizontal:N \l_@@_right_delim_dim
3392       \skip_horizontal:N \l_@@_right_margin_dim
3393       \skip_horizontal:N \l_@@_extra_right_margin_dim
3394       \@@_node_for_cell:
3395     }
3396   }
3397   \bool_gset_false:N \g_@@_empty_cell_bool
3398 }
3399 }

```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}` but, in fact, there is a flag `\g_@@_NiceArray_bool`. In `{NiceArrayWithDelims}`, some special code will be executed if this flag is raised.

```

3400 \NewDocumentEnvironment { NiceArray } { }
3401 {
3402   \bool_gset_true:N \g_@@_NiceArray_bool
3403   \str_if_empty:NT \g_@@_name_env_str
3404   { \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_@@_NiceArray_bool` is raised).

```

3405   \NiceArrayWithDelims . .
3406 }
3407 { \endNiceArrayWithDelims }

```

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

```

3408 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3409 {
3410   \NewDocumentEnvironment { #1 NiceArray } { }
3411   {
3412     \bool_gset_false:N \g_@@_NiceArray_bool
3413     \str_if_empty:NT \g_@@_name_env_str
3414     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3415     \@@_test_if_math_mode:
3416     \NiceArrayWithDelims #2 #3
3417   }
3418   { \endNiceArrayWithDelims }
3419 }
3420 \@@_def_env:nnn p ( )
3421 \@@_def_env:nnn b [ ]
3422 \@@_def_env:nnn B \{ \}
3423 \@@_def_env:nnn v | |
3424 \@@_def_env:nnn V \| \|

```

## 13 The environment {NiceMatrix} and its variants

```

3425 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3426 {
3427   \bool_set_true:N \l_@@_Matrix_bool
3428   \use:c { #1 NiceArray }
3429   {
3430     *
3431     {
3432       \int_case:nnF \l_@@_last_col_int
3433       {
3434         { -2 } { \c@MaxMatrixCols }
3435         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3436       }
3437       { \int_eval:n { \l_@@_last_col_int - 1 } }
3438     }
3439     { #2 }
3440   }
3441 }
3442 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3443 \clist_map_inline:nn { p , b , B , v , V }
3444 {
3445   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3446   {
3447     \bool_gset_false:N \g_@@_NiceArray_bool
3448     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3449     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3450     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3451   }
3452   { \use:c { end #1 NiceArray } }
3453 }

```

We define also an environment {NiceMatrix}

```

3454 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3455 {
3456   \bool_gset_false:N \g_@@_NiceArray_bool
3457   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3458   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3459   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3460 }
3461 { \endNiceArray }

```

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

```

3462 \cs_new_protected:Npn \@@_NotEmpty:
3463 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3464 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3465 {

```

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.

```

3466   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3467   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3468   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3469   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3470   \int_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim

```

```

3471 {
3472   \bool_if:NT \l_@@_hvlines_bool
3473   {
3474     \bool_set_true:N \l_@@_except_borders_bool
3475     % we should try to be more efficient in the number of lines of code here
3476     \tl_if_empty:NTF \l_@@_rules_color_tl
3477     {
3478       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3479       {
3480         \@@_stroke_block:nnn
3481         { rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim }
3482         { 1-1 }
3483         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3484       }
3485     }
3486     {
3487       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3488       {
3489         \@@_stroke_block:nnn
3490         {
3491           rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3492           draw = \l_@@_rules_color_tl
3493         }
3494         { 1-1 }
3495         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3496       }
3497     }
3498   }
3499 }
3500 \tl_if_empty:NF \l_@@_short_caption_tl
3501 {
3502   \tl_if_empty:NT \l_@@_caption_tl
3503   {
3504     \@@_error_or_warning:n { short-caption~without~caption }
3505     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3506   }
3507 }
3508 \tl_if_empty:NF \l_@@_label_tl
3509 {
3510   \tl_if_empty:NT \l_@@_caption_tl
3511   { \@@_error_or_warning:n { label~without~caption } }
3512 }
3513 \NewDocumentEnvironment { TabularNote } { b }
3514 {
3515   \bool_if:NTF \l_@@_in_code_after_bool
3516   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3517   {
3518     \tl_if_empty:NF \g_@@_tabularnote_tl
3519     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3520     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3521   }
3522 }
3523 { }
3524 \bool_set_true:N \l_@@_NiceTabular_bool
3525 \NiceArray { #2 }
3526 }
3527 { \endNiceArray }

3528 \cs_set_protected:Npn \@@_newcolumnntype #1
3529 {
3530   \cs_if_free:cT { NC @ find @ #1 }
3531   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3532   \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }

```

```

3533 \peek_meaning:NTF [
3534   { \newcol@ #1 }
3535   { \newcol@ #1 [ 0 ] }
3536 ]

```

```

3537 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3538 {

```

The following code prevents the expansion of the ‘X’ columns with the definition of that columns in `tabularx` (this would result in an error in `{NiceTabularX}`).

```

3539 \IfPackageLoadedTF { tabularx }
3540   { \newcolumnntype { X } { @@_X } }
3541   { }
3542 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3543 \dim_zero_new:N \l_@@_width_dim
3544 \dim_set:Nn \l_@@_width_dim { #1 }
3545 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3546 \bool_set_true:N \l_@@_NiceTabular_bool
3547 \NiceArray { #3 }
3548 }
3549 { \endNiceArray }

```

```

3550 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3551 {
3552   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3553   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3554   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3555   \bool_set_true:N \l_@@_NiceTabular_bool
3556   \NiceArray { #3 }
3557 }
3558 { \endNiceArray }

```

## 15 After the construction of the array

```

3559 \cs_new_protected:Npn \@@_after_array:
3560 {
3561   \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.

```

3562 \bool_if:NT \g_@@_last_col_found_bool
3563   { \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`.

```

3564 \bool_if:NT \l_@@_last_col_without_value_bool
3565   { \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.

```

3566 \bool_if:NT \l_@@_last_row_without_value_bool
3567   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3568 \tl_gput_right:Nx \g_@@_aux_tl
3569 {
3570   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3571   {
3572     \int_use:N \l_@@_first_row_int ,
3573     \int_use:N \c@iRow ,

```



```

3574         \int_use:N \g_@@_row_total_int ,
3575         \int_use:N \l_@@_first_col_int ,
3576         \int_use:N \c@jCol ,
3577         \int_use:N \g_@@_col_total_int
3578     }
3579 }

```

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`).

```

3580 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3581 {
3582     \tl_gput_right:Nx \g_@@_aux_tl
3583     {
3584         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3585         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3586     }
3587 }
3588 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3589 {
3590     \tl_gput_right:Nx \g_@@_aux_tl
3591     {
3592         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3593         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3594         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3595         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3596     }
3597 }

```

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

```

3598 \@@_create_diag_nodes:

```

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

```

3599 \pgfpicture
3600 \int_step_inline:nn \c@iRow
3601 {
3602     \pgfnodealias
3603     { \@@_env: - ##1 - last }
3604     { \@@_env: - ##1 - \int_use:N \c@jCol }
3605 }
3606 \int_step_inline:nn \c@jCol
3607 {
3608     \pgfnodealias
3609     { \@@_env: - last - ##1 }
3610     { \@@_env: - \int_use:N \c@iRow - ##1 }
3611 }
3612 \str_if_empty:NF \l_@@_name_str
3613 {
3614     \int_step_inline:nn \c@iRow
3615     {
3616         \pgfnodealias
3617         { \l_@@_name_str - ##1 - last }
3618         { \@@_env: - ##1 - \int_use:N \c@jCol }
3619     }
3620     \int_step_inline:nn \c@jCol
3621     {
3622         \pgfnodealias
3623         { \l_@@_name_str - last - ##1 }
3624         { \@@_env: - \int_use:N \c@iRow - ##1 }
3625     }
3626 }
3627 \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.

```

3628   \bool_if:NT \l_@@_parallelize_diags_bool
3629   {
3630       \int_gzero_new:N \g_@@_ddots_int
3631       \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.

```

3632       \dim_gzero_new:N \g_@@_delta_x_one_dim
3633       \dim_gzero_new:N \g_@@_delta_y_one_dim
3634       \dim_gzero_new:N \g_@@_delta_x_two_dim
3635       \dim_gzero_new:N \g_@@_delta_y_two_dim
3636   }
3637   \int_zero_new:N \l_@@_initial_i_int
3638   \int_zero_new:N \l_@@_initial_j_int
3639   \int_zero_new:N \l_@@_final_i_int
3640   \int_zero_new:N \l_@@_final_j_int
3641   \bool_set_false:N \l_@@_initial_open_bool
3642   \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.

```

3643   \bool_if:NT \l_@@_small_bool
3644   {
3645       \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3646       \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.

```

3647       \dim_set:Nn \l_@@_xdots_shorten_start_dim
3648       { 0.6 \l_@@_xdots_shorten_start_dim }
3649       \dim_set:Nn \l_@@_xdots_shorten_end_dim
3650       { 0.6 \l_@@_xdots_shorten_end_dim }
3651   }

```

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

```

3652   \@@_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.

```

3653   \@@_compute_corners:

```

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

```

3654   \@@_adjust_pos_of_blocks_seq:
3655   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3656   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3657   \IfPackageLoadedTF { tikz }
3658   {
3659       \tikzset
3660       {

```

---

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

```

3661         every-picture / .style =
3662         {
3663             overlay ,
3664             remember-picture ,
3665             name-prefix = \@@_env: -
3666         }
3667     }
3668 }
3669 { }
3670 \cs_set_eq:NN \ialign \@@_old_ialign:
3671 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3672 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3673 \cs_set_eq:NN \OverBrace \@@_OverBrace
3674 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3675 \cs_set_eq:NN \line \@@_line
3676 \g_@@_pre_code_after_tl
3677 \tl_gclear:N \g_@@_pre_code_after_tl

```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it's possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That's why we set `\CodeAfter` to be *no-op* now.

```

3678 \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).

```

3679 \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`).

```

3680 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3681 { \@@_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:`.

```

3682 \bool_set_true:N \l_@@_in_code_after_bool
3683 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3684 \scan_stop:
3685 \tl_gclear:N \g_nicematrix_code_after_tl
3686 \group_end:

```

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

```

3687 \tl_if_empty:N \g_@@_pre_code_before_tl
3688 {
3689     \tl_gput_right:Nx \g_@@_aux_tl
3690     {
3691         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3692         { \exp_not:V \g_@@_pre_code_before_tl }
3693     }
3694     \tl_gclear:N \g_@@_pre_code_before_tl
3695 }
3696 \tl_if_empty:N \g_nicematrix_code_before_tl
3697 {
3698     \tl_gput_right:Nx \g_@@_aux_tl
3699     {
3700         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3701         { \exp_not:V \g_nicematrix_code_before_tl }
3702     }
3703     \tl_gclear:N \g_nicematrix_code_before_tl
3704 }
3705 \str_gclear:N \g_@@_name_env_str

```

```
3706 \@@_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`.

```
3707 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3708 }
```

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`.

```
3709 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3710 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }
```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format  $i-j$ . However, the user is allowed to omit  $i$  or  $j$  (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It's possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```
3711 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3712 {
3713   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3714   { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3715 }
```

The following command must *not* be protected.

```
3716 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3717 {
3718   { #1 }
3719   { #2 }
3720   {
3721     \int_compare:nNnTF { #3 } > { 99 }
3722     { \int_use:N \c@iRow }
3723     { #3 }
3724   }
3725   {
3726     \int_compare:nNnTF { #4 } > { 99 }
3727     { \int_use:N \c@jCol }
3728     { #4 }
3729   }
3730   { #5 }
3731 }
```

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).

```
3732 \hook_gput_code:nnn { begindocument } { . }
3733 {
3734   \cs_new_protected:Npx \@@_draw_dotted_lines:
3735   {
3736     \c_@@_pgfortikzpicture_tl
3737     \@@_draw_dotted_lines_i:
3738     \c_@@_endpgfortikzpicture_tl
3739   }
3740 }
```

---

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

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

```

3741 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3742 {
3743   \pgfrememberpicturepositiononpagetrue
3744   \pgf@relevantforpicturesizefalse
3745   \g_@@_HVdotsfor_lines_tl
3746   \g_@@_Vdots_lines_tl
3747   \g_@@_Ddots_lines_tl
3748   \g_@@_Iddots_lines_tl
3749   \g_@@_Cdots_lines_tl
3750   \g_@@_Ldots_lines_tl
3751 }

3752 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3753 {
3754   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3755   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3756 }

```

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

```

3757 \pgfdeclareshape { @@_diag_node }
3758 {
3759   \savedanchor { \five }
3760   {
3761     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3762     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3763   }
3764   \anchor { 5 } { \five }
3765   \anchor { center } { \pgfpointorigin }
3766 }

```

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).

```

3767 \cs_new_protected:Npn \@@_create_diag_nodes:
3768 {
3769   \pgfpicture
3770   \pgfrememberpicturepositiononpagetrue
3771   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3772   {
3773     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3774     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3775     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3776     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3777     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3778     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3779     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3780     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3781     \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.

```

3782     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3783     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3784     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3785     \str_if_empty:NF \l_@@_name_str
3786     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3787   }

```

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

```

3788   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }

```

```

3789 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3790 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3791 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3792 \pgfcoordinate
3793 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3794 \pgfnodealias
3795 { \@@_env: - last }
3796 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3797 \str_if_empty:NF \l_@@_name_str
3798 {
3799   \pgfnodealias
3800   { \l_@@_name_str - \int_use:N \l_tmpa_int }
3801   { \@@_env: - \int_use:N \l_tmpa_int }
3802   \pgfnodealias
3803   { \l_@@_name_str - last }
3804   { \@@_env: - last }
3805 }
3806 \endpgfpicture
3807 }

```

## 16 We draw the dotted lines

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

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

The command `\@@_find_extremities_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.

```

3808 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3809 {

```

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

```

3810 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

3811 \int_set:Nn \l_@@_initial_i_int { #1 }
3812 \int_set:Nn \l_@@_initial_j_int { #2 }
3813 \int_set:Nn \l_@@_final_i_int { #1 }
3814 \int_set:Nn \l_@@_final_j_int { #2 }

```

We will do two loops: one when determinating the initial cell and the other when determinating 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.

```

3815 \bool_set_false:N \l_@@_stop_loop_bool
3816 \bool_do_until:Nn \l_@@_stop_loop_bool
3817 {
3818   \int_add:Nn \l_@@_final_i_int { #3 }
3819   \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3820 \bool_set_false:N \l_@@_final_open_bool
3821 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3822 {
3823   \int_compare:nNnTF { #3 } = 1
3824   { \bool_set_true:N \l_@@_final_open_bool }
3825   {
3826     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3827     { \bool_set_true:N \l_@@_final_open_bool }
3828   }
3829 }
3830 {
3831   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3832   {
3833     \int_compare:nNnT { #4 } = { -1 }
3834     { \bool_set_true:N \l_@@_final_open_bool }
3835   }
3836   {
3837     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3838     {
3839       \int_compare:nNnT { #4 } = 1
3840       { \bool_set_true:N \l_@@_final_open_bool }
3841     }
3842   }
3843 }
3844 \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.

```

3845 {

```

We do a step backwards.

```

3846 \int_sub:Nn \l_@@_final_i_int { #3 }
3847 \int_sub:Nn \l_@@_final_j_int { #4 }
3848 \bool_set_true:N \l_@@_stop_loop_bool
3849 }

```

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`.

```

3850 {
3851   \cs_if_exist:cTF
3852   {
3853     @@ _ dotted _
3854     \int_use:N \l_@@_final_i_int -
3855     \int_use:N \l_@@_final_j_int
3856   }
3857   {
3858     \int_sub:Nn \l_@@_final_i_int { #3 }
3859     \int_sub:Nn \l_@@_final_j_int { #4 }
3860     \bool_set_true:N \l_@@_final_open_bool
3861     \bool_set_true:N \l_@@_stop_loop_bool
3862   }
3863   {
3864     \cs_if_exist:cTF
3865     {
3866       pgf @ sh @ ns @ \@@_env:
3867       - \int_use:N \l_@@_final_i_int

```

```

3868         - \int_use:N \l_@@_final_j_int
3869     }
3870     { \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.

```

3871     {
3872         \cs_set:cpn
3873         {
3874             @@ _ dotted _
3875             \int_use:N \l_@@_final_i_int -
3876             \int_use:N \l_@@_final_j_int
3877         }
3878         { }
3879     }
3880 }
3881 }
3882 }

```

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

```

3883 \bool_set_false:N \l_@@_stop_loop_bool
3884 \bool_do_until:Nn \l_@@_stop_loop_bool
3885 {
3886     \int_sub:Nn \l_@@_initial_i_int { #3 }
3887     \int_sub:Nn \l_@@_initial_j_int { #4 }
3888     \bool_set_false:N \l_@@_initial_open_bool
3889     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3890     {
3891         \int_compare:nNnTF { #3 } = 1
3892         { \bool_set_true:N \l_@@_initial_open_bool }
3893         {
3894             \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3895             { \bool_set_true:N \l_@@_initial_open_bool }
3896         }
3897     }
3898     {
3899         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3900         {
3901             \int_compare:nNnT { #4 } = 1
3902             { \bool_set_true:N \l_@@_initial_open_bool }
3903         }
3904         {
3905             \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3906             {
3907                 \int_compare:nNnT { #4 } = { -1 }
3908                 { \bool_set_true:N \l_@@_initial_open_bool }
3909             }
3910         }
3911     }
3912     \bool_if:NnTF \l_@@_initial_open_bool
3913     {
3914         \int_add:Nn \l_@@_initial_i_int { #3 }
3915         \int_add:Nn \l_@@_initial_j_int { #4 }
3916         \bool_set_true:N \l_@@_stop_loop_bool
3917     }
3918     {
3919         \cs_if_exist:cTF
3920         {

```



```

3921         @@ _ dotted _
3922         \int_use:N \l_@@_initial_i_int -
3923         \int_use:N \l_@@_initial_j_int
3924     }
3925     {
3926         \int_add:Nn \l_@@_initial_i_int { #3 }
3927         \int_add:Nn \l_@@_initial_j_int { #4 }
3928         \bool_set_true:N \l_@@_initial_open_bool
3929         \bool_set_true:N \l_@@_stop_loop_bool
3930     }
3931     {
3932         \cs_if_exist:cTF
3933         {
3934             pgf @ sh @ ns @ \@@_env:
3935             - \int_use:N \l_@@_initial_i_int
3936             - \int_use:N \l_@@_initial_j_int
3937         }
3938         { \bool_set_true:N \l_@@_stop_loop_bool }
3939         {
3940             \cs_set:cpn
3941             {
3942                 @@ _ dotted _
3943                 \int_use:N \l_@@_initial_i_int -
3944                 \int_use:N \l_@@_initial_j_int
3945             }
3946             { }
3947         }
3948     }
3949 }
3950 }

```

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.

```

3951     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3952     {
3953         { \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`.

```

3954         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3955         { \int_use:N \l_@@_final_i_int }
3956         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3957         { } % for the name of the block
3958     }
3959 }

```

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).

```

3960 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3961 {
3962     \int_set:Nn \l_@@_row_min_int 1
3963     \int_set:Nn \l_@@_col_min_int 1
3964     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3965     \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`.

```

3966     \seq_map_inline:Nn \g_@@_submatrix_seq
3967     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
3968 }

```

#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.

```

3969 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3970 {
3971   \bool_if:nT
3972   {
3973     \int_compare_p:n { #3 <= #1 }
3974     && \int_compare_p:n { #1 <= #5 }
3975     && \int_compare_p:n { #4 <= #2 }
3976     && \int_compare_p:n { #2 <= #6 }
3977   }
3978   {
3979     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3980     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3981     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3982     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3983   }
3984 }

3985 \cs_new_protected:Npn \@@_set_initial_coords:
3986 {
3987   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3988   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3989 }
3990 \cs_new_protected:Npn \@@_set_final_coords:
3991 {
3992   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3993   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3994 }
3995 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3996 {
3997   \pgfpointanchor
3998   {
3999     \@@_env:
4000     - \int_use:N \l_@@_initial_i_int
4001     - \int_use:N \l_@@_initial_j_int
4002   }
4003   { #1 }
4004   \@@_set_initial_coords:
4005 }
4006 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4007 {
4008   \pgfpointanchor
4009   {
4010     \@@_env:
4011     - \int_use:N \l_@@_final_i_int
4012     - \int_use:N \l_@@_final_j_int
4013   }
4014   { #1 }
4015   \@@_set_final_coords:
4016 }

4017 \cs_new_protected:Npn \@@_open_x_initial_dim:
4018 {
4019   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4020   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4021   {
4022     \cs_if_exist:cT
4023     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4024     {
4025       \pgfpointanchor
4026       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4027       { west }

```

```

4028         \dim_set:Nn \l_@@_x_initial_dim
4029         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4030     }
4031 }

```

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

```

4032 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4033 {
4034     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4035     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4036     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4037 }
4038 }
4039 \cs_new_protected:Npn \@@_open_x_final_dim:
4040 {
4041     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4042     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4043     {
4044         \cs_if_exist:cT
4045         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4046         {
4047             \pgfpointanchor
4048             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4049             { east }
4050             \dim_set:Nn \l_@@_x_final_dim
4051             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4052         }
4053     }

```

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

```

4054 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4055 {
4056     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4057     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4058     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4059 }
4060 }

```

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.

```

4061 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4062 {
4063     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4064     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4065     {
4066         \@@_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.

```

4067 \group_begin:
4068     \int_compare:nNnTF { #1 } = 0
4069     { \color { nicematrix-first-row } }
4070     {

```

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.

```

4071         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4072         { \color { nicematrix-last-row } }
4073     }
4074     \keys_set:nn { NiceMatrix / xdots } { #3 }
4075     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4076     \@@_actually_draw_Ldots:
4077 \group_end:

```

```

4078     }
4079 }

```

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`.

```

4080 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4081 {
4082   \bool_if:NTF \l_@@_initial_open_bool
4083   {
4084     \@@_open_x_initial_dim:
4085     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4086     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4087   }
4088   { \@@_set_initial_coords_from_anchor:n { base-east } }
4089   \bool_if:NTF \l_@@_final_open_bool
4090   {
4091     \@@_open_x_final_dim:
4092     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4093     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4094   }
4095   { \@@_set_final_coords_from_anchor:n { base-west } }

```

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 (?).

```

4096   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4097   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4098   \@@_draw_line:
4099 }

```

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.

```

4100 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4101 {
4102   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4103   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4104   {
4105     \@@_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.

```

4106   \group_begin:
4107   \int_compare:nNnTF { #1 } = 0
4108   { \color { nicematrix-first-row } }
4109   {

```

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.

```

4110     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4111     { \color { nicematrix-last-row } }
4112   }
4113   \keys_set:nn { NiceMatrix / xdots } { #3 }

```

```

4114         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4115         \@@_actually_draw_Cdots:
4116     \group_end:
4117 }
4118 }

```

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.`

```

4119 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4120 {
4121     \bool_if:NTF \l_@@_initial_open_bool
4122     { \@@_open_x_initial_dim: }
4123     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4124     \bool_if:NTF \l_@@_final_open_bool
4125     { \@@_open_x_final_dim: }
4126     { \@@_set_final_coords_from_anchor:n { mid-west } }
4127     \bool_lazy_and:nnTF
4128     \l_@@_initial_open_bool
4129     \l_@@_final_open_bool
4130     {
4131         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4132         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4133         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4134         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4135         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4136     }
4137     {
4138         \bool_if:NT \l_@@_initial_open_bool
4139         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4140         \bool_if:NT \l_@@_final_open_bool
4141         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4142     }
4143     \@@_draw_line:
4144 }
4145 \cs_new_protected:Npn \@@_open_y_initial_dim:
4146 {
4147     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4148     \dim_set:Nn \l_@@_y_initial_dim
4149     {
4150         \fp_to_dim:n
4151         {
4152             \pgf@y
4153             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4154         }
4155     } % modified 6.13c
4156     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4157     {
4158         \cs_if_exist:cT
4159         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4160         {
4161             \pgfpointanchor
4162             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4163             { north }

```

```

4164         \dim_set:Nn \l_@@_y_initial_dim
4165         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4166     }
4167 }
4168 }
4169 \cs_new_protected:Npn \@@_open_y_final_dim:
4170 {
4171     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4172     \dim_set:Nn \l_@@_y_final_dim
4173     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4174     % modified 6.13c
4175     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4176     {
4177         \cs_if_exist:cT
4178         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4179         {
4180             \pgfpointanchor
4181             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4182             { south }
4183             \dim_set:Nn \l_@@_y_final_dim
4184             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4185         }
4186     }
4187 }

```

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.

```

4188 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4189 {
4190     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4191     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4192     {
4193         \@@_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.

```

4194     \group_begin:
4195     \int_compare:nNnTF { #2 } = 0
4196     { \color { nicematrix-first-col } }
4197     {
4198         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4199         { \color { nicematrix-last-col } }
4200     }
4201     \keys_set:nn { NiceMatrix / xdots } { #3 }
4202     \tl_if_empty:VF \l_@@_xdots_color_tl
4203     { \color { \l_@@_xdots_color_tl } }
4204     \@@_actually_draw_Vdots:
4205     \group_end:
4206 }
4207 }

```

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`.

```
4208 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4209 {
```

The boolean `\l_tmpa_bool` indicates whether the column is of type `l` or may be considered as if.

```
4210 \bool_set_false:N \l_tmpa_bool
```

First the case when the line is closed on both ends.

```
4211 \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
4212 {
4213   \@@_set_initial_coords_from_anchor:n { south-west }
4214   \@@_set_final_coords_from_anchor:n { north-west }
4215   \bool_set:Nn \l_tmpa_bool
4216   { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4217 }
```

Now, we try to determine whether the column is of type `c` or may be considered as if.

```
4218 \bool_if:NTF \l_@@_initial_open_bool
4219 \@@_open_y_initial_dim:
4220 { \@@_set_initial_coords_from_anchor:n { south } }
4221 \bool_if:NTF \l_@@_final_open_bool
4222 \@@_open_y_final_dim:
4223 { \@@_set_final_coords_from_anchor:n { north } }
4224 \bool_if:NTF \l_@@_initial_open_bool
4225 {
4226   \bool_if:NTF \l_@@_final_open_bool
4227   {
4228     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4229     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4230     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4231     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4232     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4233   }
```

We may think that the final user won't use a "last column" which contains only a command `\Vdots`. However, if the `\Vdots` is in fact used to draw, not a dotted line, but an arrow (to indicate the number of rows of the matrix), it may be really encountered.

```
4233 \int_compare:nNnT \l_@@_last_col_int > { -2 }
4234 {
4235   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
4236   {
4237     \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
4238     \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
4239     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4240     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
4241   }
4242 }
4243 }
4244 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
4245 }
4246 {
4247   \bool_if:NTF \l_@@_final_open_bool
4248   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
4249 }
```

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.

```
4250 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4251 {
4252   \dim_set:Nn \l_@@_x_initial_dim
4253   {
4254     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4255     \l_@@_x_initial_dim \l_@@_x_final_dim
4256   }
4257   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4258 }
```

```

4259     }
4260   }
4261   \@@_draw_line:
4262 }

```

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.

```

4263 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4264 {
4265   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4266   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4267   {
4268     \@@_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.

```

4269     \group_begin:
4270     \keys_set:nn { NiceMatrix / xdots } { #3 }
4271     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4272     \@@_actually_draw_Ddots:
4273     \group_end:
4274   }
4275 }

```

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`.

```

4276 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4277 {
4278   \bool_if:NTF \l_@@_initial_open_bool
4279   {
4280     \@@_open_y_initial_dim:
4281     \@@_open_x_initial_dim:
4282   }
4283   { \@@_set_initial_coords_from_anchor:n { south-east } }
4284   \bool_if:NTF \l_@@_final_open_bool
4285   {
4286     \@@_open_x_final_dim:
4287     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4288   }
4289   { \@@_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.

```

4290   \bool_if:NT \l_@@_parallelize_diags_bool
4291   {
4292     \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).

```

4293     \int_compare:nNnTF \g_@@_ddots_int = 1

```



If the diagonal line is the first one, we have no adjustment of the line to do but we store the  $\Delta_x$  and the  $\Delta_y$  of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4294     {
4295         \dim_gset:Nn \g_@@_delta_x_one_dim
4296         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4297         \dim_gset:Nn \g_@@_delta_y_one_dim
4298         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4299     }

```

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`.

```

4300     {
4301         \dim_set:Nn \l_@@_y_final_dim
4302         {
4303             \l_@@_y_initial_dim +
4304             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4305             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4306         }
4307     }
4308 }
4309 \@@_draw_line:
4310 }

```

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.

```

4311 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4312 {
4313     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4314     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4315     {
4316         \@@_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.

```

4317     \group_begin:
4318     \keys_set:nn { NiceMatrix / xdots } { #3 }
4319     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4320     \@@_actually_draw_Iddots:
4321     \group_end:
4322 }
4323 }

```

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`.

```

4324 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4325 {
4326     \bool_if:NTF \l_@@_initial_open_bool
4327     {
4328         \@@_open_y_initial_dim:
4329         \@@_open_x_initial_dim:

```

```

4330     }
4331     { \@@_set_initial_coords_from_anchor:n { south-west } }
4332     \bool_if:NTF \l_@@_final_open_bool
4333     {
4334         \@@_open_y_final_dim:
4335         \@@_open_x_final_dim:
4336     }
4337     { \@@_set_final_coords_from_anchor:n { north-east } }
4338     \bool_if:NT \l_@@_parallelize_diags_bool
4339     {
4340         \int_gincr:N \g_@@_iddots_int
4341         \int_compare:nNnTF \g_@@_iddots_int = 1
4342         {
4343             \dim_gset:Nn \g_@@_delta_x_two_dim
4344             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4345             \dim_gset:Nn \g_@@_delta_y_two_dim
4346             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4347         }
4348         {
4349             \dim_set:Nn \l_@@_y_final_dim
4350             {
4351                 \l_@@_y_initial_dim +
4352                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4353                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4354             }
4355         }
4356     }
4357     \@@_draw_line:
4358 }

```

## 17 The actual instructions for drawing the dotted lines with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

4359 \cs_new_protected:Npn \@@_draw_line:
4360 {
4361     \pgfrememberpicturepositiononpagetrue
4362     \pgf@relevantforpicturesizefalse
4363     \bool_lazy_or:nnTF
4364     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4365     \l_@@_dotted_bool
4366     \@@_draw_standard_dotted_line:
4367     \@@_draw_unstandard_dotted_line:
4368 }

```

We have to do a special construction with `\exp_args:NV` to be able to put in the list of options in the correct place in the Tikz instruction.

```

4369 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4370 {
4371   \begin { scope }
4372   \@@_draw_unstandard_dotted_line:o
4373   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4374 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4375 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4376 {
4377   \@@_draw_unstandard_dotted_line:nVV
4378   { #1 }
4379   \l_@@_xdots_up_tl
4380   \l_@@_xdots_down_tl
4381 }
4382 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4383 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
4384 {
4385   \draw
4386   [
4387     #1 ,
4388     shorten-> = \l_@@_xdots_shorten_end_dim ,
4389     shorten-< = \l_@@_xdots_shorten_start_dim ,
4390   ]
4391   ( \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).

```

4392   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
4393   node [ sloped , below ] { $ \scriptstyle #3 $ }
4394   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4395   \end { scope }
4396 }
4397 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnn { n V V }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real round dots).

```

4398 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4399 {
4400   \bool_lazy_and:nnF
4401   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4402   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4403   {
4404     \pgfscope
4405     \pgftransformshift
4406     {
4407       \pgfpointlineattime { 0.5 }
4408       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4409       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4410     }
4411     \pgftransformrotate
4412     {
4413       \fp_eval:n
4414       {
4415         atand
4416         (
4417           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4418           \l_@@_x_final_dim - \l_@@_x_initial_dim

```

```

4419         )
4420     }
4421 }
4422 \pgfnode
4423 { rectangle }
4424 { south }
4425 {
4426     \c_math_toggle_token
4427     \scriptstyle \l_@@_xdots_up_tl
4428     \c_math_toggle_token
4429 }
4430 { }
4431 { \pgfusepath { } }
4432 \pgfnode
4433 { rectangle }
4434 { north }
4435 {
4436     \c_math_toggle_token
4437     \scriptstyle \l_@@_xdots_down_tl
4438     \c_math_toggle_token
4439 }
4440 { }
4441 { \pgfusepath { } }
4442 \endpgfscope
4443 }
4444 \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.

```

4445     \dim_zero_new:N \l_@@_l_dim
4446     \dim_set:Nn \l_@@_l_dim
4447     {
4448         \fp_to_dim:n
4449         {
4450             sqrt
4451             (
4452                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4453                 +
4454                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4455             )
4456         }
4457     }

```

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.

```

4458     \bool_lazy_or:nnF
4459     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4460     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4461     \@@_draw_standard_dotted_line_i:
4462 \group_end:
4463 }
4464 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4465 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4466 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4467     \bool_if:NTF \l_@@_initial_open_bool
4468     {
4469         \bool_if:NTF \l_@@_final_open_bool
4470         {
4471             \int_set:Nn \l_tmpa_int

```

```

4472         { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
4473     }
4474     {
4475         \int_set:Nn \l_tmpa_int
4476         {
4477             \dim_ratio:nn
4478             { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
4479             \l_@@_xdots_inter_dim
4480         }
4481     }
4482 }
4483 {
4484     \bool_if:NTF \l_@@_final_open_bool
4485     {
4486         \int_set:Nn \l_tmpa_int
4487         {
4488             \dim_ratio:nn
4489             { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
4490             \l_@@_xdots_inter_dim
4491         }
4492     }
4493     {
4494         \int_set:Nn \l_tmpa_int
4495         {
4496             \dim_ratio:nn
4497             {
4498                 \l_@@_l_dim
4499                 - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4500             }
4501             \l_@@_xdots_inter_dim
4502         }
4503     }
4504 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4505     \dim_set:Nn \l_tmpa_dim
4506     {
4507         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4508         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4509     }
4510     \dim_set:Nn \l_tmpb_dim
4511     {
4512         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4513         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4514     }

```

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.

```

4515     \dim_gadd:Nn \l_@@_x_initial_dim
4516     {
4517         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4518         \dim_ratio:nn
4519         {
4520             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4521             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4522         }
4523         { 2 \l_@@_l_dim }
4524     }
4525     \dim_gadd:Nn \l_@@_y_initial_dim
4526     {
4527         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4528         \dim_ratio:nn
4529         {

```

```

4530         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4531         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4532     }
4533     { 2 \l_@@_l_dim }
4534 }
4535 \pgf@relevantforpicturesizefalse
4536 \int_step_inline:nnn 0 \l_tmpa_int
4537 {
4538     \pgfpathcircle
4539     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4540     { \l_@@_xdots_radius_dim }
4541     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4542     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4543 }
4544 \pgfusepathqfill
4545 }

```

## 18 User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4546 \hook_gput_code:nnn { begindocument } { . }
4547 {
4548     \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
4549     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4550     \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4551     {
4552         \int_compare:nNnTF \c@jCol = 0
4553         { \@@_error:nn { in~first~col } \Ldots }
4554         {
4555             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4556             { \@@_error:nn { in~last~col } \Ldots }
4557             {
4558                 \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4559                 { #1 , down = #2 , up = #3 }
4560             }
4561         }
4562         \bool_if:NF \l_@@_nullify_dots_bool
4563         { \phantom { \ensuremath { \@@_old_ldots } } }
4564         \bool_gset_true:N \g_@@_empty_cell_bool
4565     }

4566     \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4567     {
4568         \int_compare:nNnTF \c@jCol = 0
4569         { \@@_error:nn { in~first~col } \Cdots }
4570         {
4571             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4572             { \@@_error:nn { in~last~col } \Cdots }
4573             {
4574                 \@@_instruction_of_type:nnn \c_false_bool { Cdots }

```

```

4575         { #1 , down = #2 , up = #3 }
4576     }
4577 }
4578 \bool_if:NF \l_@@_nullify_dots_bool
4579 { \phantom { \ensuremath { \@@_old_cdots } } }
4580 \bool_gset_true:N \g_@@_empty_cell_bool
4581 }

4582 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4583 {
4584     \int_compare:nNnTF \c@iRow = 0
4585     { \@@_error:nn { in~first~row } \Vdots }
4586     {
4587         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4588         { \@@_error:nn { in~last~row } \Vdots }
4589         {
4590             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4591             { #1 , down = #2 , up = #3 }
4592         }
4593     }
4594     \bool_if:NF \l_@@_nullify_dots_bool
4595     { \phantom { \ensuremath { \@@_old_vdots } } }
4596     \bool_gset_true:N \g_@@_empty_cell_bool
4597 }

4598 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4599 {
4600     \int_case:nnF \c@iRow
4601     {
4602         0 { \@@_error:nn { in~first~row } \Ddots }
4603         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4604     }
4605     {
4606         \int_case:nnF \c@jCol
4607         {
4608             0 { \@@_error:nn { in~first~col } \Ddots }
4609             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4610         }
4611         {
4612             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4613             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4614             { #1 , down = #2 , up = #3 }
4615         }
4616     }
4617 }
4618 \bool_if:NF \l_@@_nullify_dots_bool
4619 { \phantom { \ensuremath { \@@_old_ddots } } }
4620 \bool_gset_true:N \g_@@_empty_cell_bool
4621 }

4622 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4623 {
4624     \int_case:nnF \c@iRow
4625     {
4626         0 { \@@_error:nn { in~first~row } \Iddots }
4627         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4628     }
4629     {
4630         \int_case:nnF \c@jCol
4631         {
4632             0 { \@@_error:nn { in~first~col } \Iddots }

```

```

4633         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4634     }
4635     {
4636         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4637         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4638         { #1 , down = #2 , up = #3 }
4639     }
4640 }
4641 \bool_if:NF \l_@@_nullify_dots_bool
4642 { \phantom { \ensuremath { \@@_old_iddots } } }
4643 \bool_gset_true:N \g_@@_empty_cell_bool
4644 }
4645 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

4646 \keys_define:nn { NiceMatrix / Ddots }
4647 {
4648     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4649     draw-first .default:n = true ,
4650     draw-first .value_forbidden:n = true
4651 }

```

The command `\@@_Hspace`: will be linked to `\hspace` in `{NiceArray}`.

```

4652 \cs_new_protected:Npn \@@_Hspace:
4653 {
4654     \bool_gset_true:N \g_@@_empty_cell_bool
4655     \hspace
4656 }

```

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`.

```

4657 \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`.

```

4658 \cs_new:Npn \@@_Hdotsfor:
4659 {
4660     \bool_lazy_and:nnTF
4661     { \int_compare_p:nNn \c@jCol = 0 }
4662     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4663     {
4664         \bool_if:NTF \g_@@_after_col_zero_bool
4665         {
4666             \multicolumn { 1 } { c } { }
4667             \@@_Hdotsfor_i
4668         }
4669         { \@@_fatal:n { Hdotsfor~in~col~0 } }
4670     }
4671     {
4672         \multicolumn { 1 } { c } { }
4673         \@@_Hdotsfor_i
4674     }
4675 }

```

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:`).

```

4676 \hook_gput_code:nnn { begindocument } { . }

```



```

4677 {
4678   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4679   \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.

```

4680   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4681   {
4682     \tl_gput_right:Nx \g_@@_Hvdotsfor_lines_tl
4683     {
4684       \@@_Hdotsfor:nnnn
4685       { \int_use:N \c@iRow }
4686       { \int_use:N \c@jCol }
4687       { #2 }
4688       {
4689         #1 , #3 ,
4690         down = \exp_not:n { #4 } ,
4691         up = \exp_not:n { #5 }
4692       }
4693     }
4694     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4695   }
4696 }

```

```

4697 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4698 {
4699   \bool_set_false:N \l_@@_initial_open_bool
4700   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4701   \int_set:Nn \l_@@_initial_i_int { #1 }
4702   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

4703   \int_compare:nNnTF { #2 } = 1
4704   {
4705     \int_set:Nn \l_@@_initial_j_int 1
4706     \bool_set_true:N \l_@@_initial_open_bool
4707   }
4708   {
4709     \cs_if_exist:cTF
4710     {
4711       pgf @ sh @ ns @ \@@_env:
4712       - \int_use:N \l_@@_initial_i_int
4713       - \int_eval:n { #2 - 1 }
4714     }
4715     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4716     {
4717       \int_set:Nn \l_@@_initial_j_int { #2 }
4718       \bool_set_true:N \l_@@_initial_open_bool
4719     }
4720   }
4721   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4722   {
4723     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4724     \bool_set_true:N \l_@@_final_open_bool
4725   }
4726   {
4727     \cs_if_exist:cTF
4728     {
4729       pgf @ sh @ ns @ \@@_env:
4730       - \int_use:N \l_@@_final_i_int
4731       - \int_eval:n { #2 + #3 }
4732     }

```

```

4733         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4734     {
4735         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4736         \bool_set_true:N \l_@@_final_open_bool
4737     }
4738 }
4739 \group_begin:
4740 \int_compare:nNnTF { #1 } = 0
4741 { \color { nicematrix-first-row } }
4742 {
4743     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4744     { \color { nicematrix-last-row } }
4745 }
4746 \keys_set:nn { NiceMatrix / xdots } { #4 }
4747 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4748 \@@_actually_draw_Ldots:
4749 \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).

```

4750     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4751     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4752 }
4753 \hook_gput_code:nnn { begindocument } { . }
4754 {
4755     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4756     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4757     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4758     {
4759         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4760         {
4761             \@@_Vdotsfor:nnnn
4762             { \int_use:N \c@iRow }
4763             { \int_use:N \c@jCol }
4764             { #2 }
4765             {
4766                 #1 , #3 ,
4767                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4768             }
4769         }
4770     }
4771 }

```

Enf of `\AddToHook`.

```

4772 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4773 {
4774     \bool_set_false:N \l_@@_initial_open_bool
4775     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

4776     \int_set:Nn \l_@@_initial_j_int { #2 }
4777     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it’s a bit more complicated.

```

4778     \int_compare:nNnTF #1 = 1
4779     {
4780         \int_set:Nn \l_@@_initial_i_int 1
4781         \bool_set_true:N \l_@@_initial_open_bool
4782     }
4783     {
4784         \cs_if_exist:CTF

```

```

4785     {
4786         pgf @ sh @ ns @ \@@_env:
4787         - \int_eval:n { #1 - 1 }
4788         - \int_use:N \l_@@_initial_j_int
4789     }
4790     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4791     {
4792         \int_set:Nn \l_@@_initial_i_int { #1 }
4793         \bool_set_true:N \l_@@_initial_open_bool
4794     }
4795 }
4796 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4797 {
4798     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4799     \bool_set_true:N \l_@@_final_open_bool
4800 }
4801 {
4802     \cs_if_exist:cTF
4803     {
4804         pgf @ sh @ ns @ \@@_env:
4805         - \int_eval:n { #1 + #3 }
4806         - \int_use:N \l_@@_final_j_int
4807     }
4808     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4809     {
4810         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4811         \bool_set_true:N \l_@@_final_open_bool
4812     }
4813 }
4814 \group_begin:
4815 \int_compare:nNnTF { #2 } = 0
4816 { \color { nicematrix-first-col } }
4817 {
4818     \int_compare:nNnT { #2 } = \g_@@_col_total_int
4819     { \color { nicematrix-last-col } }
4820 }
4821 \keys_set:nn { NiceMatrix / xdots } { #4 }
4822 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4823 \@@_actually_draw_Vdots:
4824 \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).

```

4825     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4826     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4827 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

4828 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

## 19 The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format  $i-j$ , our command applies the command `\int_eval:n` to  $i$  and  $j$  ;
- If not (that is to say, when it's a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).<sup>13</sup>

```

4829 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4830 {
4831   \tl_if_empty:nTF { #2 }
4832     { #1 }
4833     { \@@_double_int_eval_i:n #1-#2 \q_stop }
4834 }
4835 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
4836 { \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).

```

4837 \hook_gput_code:nnn { begindocument } { . }
4838 {
4839   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4840   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4841   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4842     {
4843       \group_begin:
4844       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4845       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4846       \use:e
4847       {
4848         \@@_line_i:nn
4849         { \@@_double_int_eval:n #2 - \q_stop }
4850         { \@@_double_int_eval:n #3 - \q_stop }
4851       }
4852       \group_end:
4853     }
4854 }
4855 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4856 {
4857   \bool_set_false:N \l_@@_initial_open_bool
4858   \bool_set_false:N \l_@@_final_open_bool
4859   \bool_if:nTF
4860     {
4861       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4862       ||
4863       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4864     }
4865     {
4866       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4867     }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

4868   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
4869 }
4870 \hook_gput_code:nnn { begindocument } { . }
4871 {
4872   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4873   {
```

---

<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`.

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:`.

```

4874     \c_@@_pgfortikzpicture_tl
4875     \@@_draw_line_iii:nn { #1 } { #2 }
4876     \c_@@_endpgfortikzpicture_tl
4877   }
4878 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

4879 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4880 {
4881   \pgfrememberpicturepositiononpagetrue
4882   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4883   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4884   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4885   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4886   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4887   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4888   \@@_draw_line:
4889 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

## 20 The command `\RowStyle`

```

4890 \keys_define:nn { NiceMatrix / RowStyle }
4891 {
4892   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4893   cell-space-top-limit .initial:n = \c_zero_dim ,
4894   cell-space-top-limit .value_required:n = true ,
4895   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4896   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4897   cell-space-bottom-limit .value_required:n = true ,
4898   cell-space-limits .meta:n =
4899   {
4900     cell-space-top-limit = #1 ,
4901     cell-space-bottom-limit = #1 ,
4902   } ,
4903   color .tl_set:N = \l_@@_color_tl ,
4904   color .value_required:n = true ,
4905   bold .bool_set:N = \l_tmpa_bool ,
4906   bold .default:n = true ,
4907   bold .initial:n = false ,
4908   nb-rows .code:n =
4909   \str_if_eq:nnTF { #1 } { * }
4910   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
4911   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
4912   nb-rows .value_required:n = true ,
4913   rowcolor .tl_set:N = \l_tmpa_tl ,
4914   rowcolor .value_required:n = true ,
4915   rowcolor .initial:n = ,
4916   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
4917 }

4918 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4919 {

```

```

4920 \group_begin:
4921 \tl_clear:N \l_tmpa_tl % value of \rowcolor
4922 \tl_clear:N \l_@@_color_tl
4923 \int_set:Nn \l_@@_key_nb_rows_int 1
4924 \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4925 \tl_if_empty:NF \l_tmpa_tl
4926 {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

4927 \tl_gput_right:Nx \g_@@_pre_code_before_tl
4928 {

```

The command `\@@_exp_color_arg:NV` is *fully expandable*.

```

4929 \@@_exp_color_arg:NV \@@_rectanglecolor \l_tmpa_tl
4930 { \int_use:N \c@iRow - \int_use:N \c@jCol }
4931 { \int_use:N \c@iRow - * }
4932 }

```

Then, the other rows (if there is several rows).

```

4933 \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4934 {
4935   \tl_gput_right:Nx \g_@@_pre_code_before_tl
4936   {
4937     \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
4938     {
4939       \int_eval:n { \c@iRow + 1 }
4940       - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4941     }
4942   }
4943 }
4944 }
4945 \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4946 \tl_gput_right:Nx \g_@@_row_style_tl
4947 { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4948 \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

4949 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4950 {
4951   \tl_gput_right:Nx \g_@@_row_style_tl
4952   {
4953     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4954     {
4955       \dim_set:Nn \l_@@_cell_space_top_limit_dim
4956       { \dim_use:N \l_tmpa_dim }
4957     }
4958   }
4959 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

4960 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4961 {
4962   \tl_gput_right:Nx \g_@@_row_style_tl
4963   {
4964     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4965     {
4966       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4967       { \dim_use:N \l_tmpb_dim }
4968     }
4969   }
4970 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

4971 \tl_if_empty:NF \l_@@_color_tl
4972 {
4973   \tl_gput_right:Nx \g_@@_row_style_tl

```

```

4974     {
4975         \mode_leave_vertical:
4976         \@@_color:n { \l_@@_color_tl }
4977     }
4978 }
\l_tmpa_bool is the value of the key bold.
4979 \bool_if:NT \l_tmpa_bool
4980 {
4981     \tl_gput_right:Nn \g_@@_row_style_tl
4982     {
4983         \if_mode_math:
4984             \c_math_toggle_token
4985             \bfseries \boldmath
4986             \c_math_toggle_token
4987         \else:
4988             \bfseries \boldmath
4989         \fi:
4990     }
4991 }
4992 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4993 \group_end:
4994 \g_@@_row_style_tl
4995 \ignorespaces
4996 }

```

## 21 Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That’s why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`—in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor`, `\@@_rectanglecolor` and `\@@_rowlistcolors` don’t directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\g_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\g_@@_color_i_tl`. In that token list, the instructions will be written using `\@@_cartesian_color:nn` and `\@@_rectanglecolor:nn`.

`#1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_colors_seq:nn` doesn’t only add a color to `\g_@@_colors_seq`: it also updates the corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```

4997 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4998 {

```

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.

```

4999     \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`.

```

5000     \str_if_in:nnF { #1 } { !! }
5001     {
5002         \seq_map_indexed_inline:Nn \g_@@_colors_seq
5003         { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5004     }
5005     \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

5006      {
5007          \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5008          \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5009      }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5010      { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5011  }

5012 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5013 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }

```

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`).

```

5014 \cs_new_protected:Npn \@@_actually_color:
5015 {
5016     \pgfpicture
5017     \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.

```

5018     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5019     {
5020         \pgfsetcornersarced
5021         {
5022             \pgfpoint
5023             { \l_@@_tab_rounded_corners_dim }
5024             { \l_@@_tab_rounded_corners_dim }
5025         }
5026     % \end{macrocode}
5027     % Because we want \pkg{nicematrix} compatible with arrays constructed by
5028     % \pkg{array}, the nodes for the rows and columns (that is to say the nodes
5029     % |row-|\textsl{i} and |col-|\textsl{j}|) have not always the expected position,
5030     % that is to say, there is sometimes a slight shifting of something such as
5031     % |\arrayrulewidth|. Now, for the clipping, we have to change slightly the
5032     % position of that clipping whether a rounded rectangle around the array is
5033     % required. That's the point which is tested in the following line.
5034     % \begin{macrocode}
5035     \bool_if:NTF \l_@@_hvlines_bool
5036     {
5037         \pgfpathrectanglecorners
5038         {
5039             \pgfpointadd
5040             { \@@_qpoint:n { row-1 } }
5041             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5042         }
5043         {
5044             \pgfpointadd
5045             {
5046                 \@@_qpoint:n
5047                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5048             }
5049             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5050         }
5051     }
5052     {
5053         \pgfpathrectanglecorners
5054         { \@@_qpoint:n { row-1 } }
5055         {
5056             \pgfpointadd
5057             {
5058                 \@@_qpoint:n

```



```

5059         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5060     }
5061     { \pgfpicture \c_zero_dim \arrayrulewidth }
5062 }
5063 }
5064 \pgfusepath { clip }
5065 }
5066 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5067 {
5068     \begin { pgfscope }
5069         \@@_color_opacity ##2
5070         \use:c { g_@@_color _ ##1 _tl }
5071         \tl_gclear:c { g_@@_color _ ##1 _tl }
5072         \pgfusepath { fill }
5073     \end { pgfscope }
5074 }
5075 \endpgfpicture
5076 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5077 \cs_new_protected:Npn \@@_color_opacity
5078 {
5079     \peek_meaning:NTF [
5080         { \@@_color_opacity:w }
5081         { \@@_color_opacity:w [ ] }
5082     }

```

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 curriification.

```

5083 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5084 {
5085     \tl_clear:N \l_tmpa_tl
5086     \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.

```

5087     \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5088     \tl_if_empty:NTF \l_tmpb_tl
5089         { \@declaredcolor }
5090         { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5091 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5092 \keys_define:nn { nicematrix / color-opacity }
5093 {
5094     opacity .tl_set:N          = \l_tmpa_tl ,
5095     opacity .value_required:n = true
5096 }

5097 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5098 {
5099     \tl_set:Nn \l_@@_rows_tl { #1 }
5100     \tl_set:Nn \l_@@_cols_tl { #2 }
5101     \@@_cartesian_path:
5102 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5103 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5104 {

```

```

5105 \tl_if_blank:nF { #2 }
5106 {
5107   \@@_add_to_colors_seq:xn
5108   { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5109   { \@@_cartesian_color:nn { #3 } { - } }
5110 }
5111 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5112 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5113 {
5114   \tl_if_blank:nF { #2 }
5115   {
5116     \@@_add_to_colors_seq:xn
5117     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5118     { \@@_cartesian_color:nn { - } { #3 } }
5119   }
5120 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5121 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5122 {
5123   \tl_if_blank:nF { #2 }
5124   {
5125     \@@_add_to_colors_seq:xn
5126     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5127     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5128   }
5129 }

```

The last argument is the radius of the corners of the rectangle.

```

5130 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5131 {
5132   \tl_if_blank:nF { #2 }
5133   {
5134     \@@_add_to_colors_seq:xn
5135     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5136     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5137   }
5138 }

```

The last argument is the radius of the corners of the rectangle.

```

5139 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5140 {
5141   \@@_cut_on_hyphen:w #1 \q_stop
5142   \tl_clear_new:N \l_@@_tmpc_tl
5143   \tl_clear_new:N \l_@@_tmpd_tl
5144   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5145   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5146   \@@_cut_on_hyphen:w #2 \q_stop
5147   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5148   \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`.

```

5149 \@@_cartesian_path:n { #3 }
5150 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5151 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5152 {

```

```

5153 \clist_map_inline:nn { #3 }
5154 { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5155 }

5156 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5157 {
5158   \int_step_inline:nn { \int_use:N \c@iRow }
5159   {
5160     \int_step_inline:nn { \int_use:N \c@jCol }
5161     {
5162       \int_if_even:nTF { ####1 + ##1 }
5163       { \@@_cellcolor [ #1 ] { #2 } }
5164       { \@@_cellcolor [ #1 ] { #3 } }
5165       { ##1 - ####1 }
5166     }
5167   }
5168 }

```

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”.

```

5169 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5170 {
5171   \@@_rectanglecolor [ #1 ] { #2 }
5172   { 1 - 1 }
5173   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5174 }

5175 \keys_define:nn { NiceMatrix / rowcolors }
5176 {
5177   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5178   respect-blocks .default:n = true ,
5179   cols .tl_set:N = \l_@@_cols_tl ,
5180   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5181   restart .default:n = true ,
5182   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5183 }

```

The command `\rowcolors` (accessible in the code-before) 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]`.

**#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*.

```

5184 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5185 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5186   \group_begin:
5187   \seq_clear_new:N \l_@@_colors_seq
5188   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5189   \tl_clear_new:N \l_@@_cols_tl
5190   \tl_set:Nn \l_@@_cols_tl { - }
5191   \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).

```

5192   \int_zero_new:N \l_@@_color_int
5193   \int_set:Nn \l_@@_color_int 1
5194   \bool_if:NT \l_@@_respect_blocks_bool
5195   {

```

We don't want to take into account a block which is completely in the “first column” of (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`).

```

5196     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5197     \seq_set_filter:Nn \l_tmpa_seq \l_tmpb_seq
5198     { \@@_not_in_exterior_p:nnnnn ##1 }
5199   }
5200   \pgfpicture
5201   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5202   \clist_map_inline:nn { #2 }
5203   {
5204     \tl_set:Nn \l_tmpa_tl { ##1 }
5205     \tl_if_in:NnTF \l_tmpa_tl { - }
5206     { \@@_cut_on_hyphen:w ##1 \q_stop }
5207     { \tl_set:Nx \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.

```

5208     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5209     \bool_if:NTF \l_@@_rowcolors_restart_bool
5210     { \int_set:Nn \l_@@_color_int 1 }
5211     { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
5212     \int_zero_new:N \l_@@_tmpc_int
5213     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5214     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5215     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5216         \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).

```

5217         \bool_if:NT \l_@@_respect_blocks_bool
5218         {
5219           \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5220           { \@@_intersect_our_row_p:nnnnn #####1 }
5221           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5222         }
5223         \tl_set:Nx \l_@@_rows_tl
5224         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5225         \tl_clear_new:N \l_@@_color_tl
5226         \tl_set:Nx \l_@@_color_tl
5227         {
5228           \@@_color_index:n
5229           {
5230             \int_mod:nn
5231             { \l_@@_color_int - 1 }
5232             { \seq_count:N \l_@@_colors_seq }
5233             + 1
5234           }
5235         }
5236         \tl_if_empty:NF \l_@@_color_tl
5237         {
5238           \@@_add_to_colors_seq:xx
5239           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5240           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5241         }
5242         \int_incr:N \l_@@_color_int
5243         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5244       }

```

```

5245     }
5246     \endpgfpicture
5247     \group_end:
5248 }

```

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.

```

5249 \cs_new:Npn \@@_color_index:n #1
5250 {
5251     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5252     { \@@_color_index:n { #1 - 1 } }
5253     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5254 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

5255 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
5256 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

5257 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5258 {
5259     \int_compare:nNnT { #3 } > \l_tmpb_int
5260     { \int_set:Nn \l_tmpb_int { #3 } }
5261 }

5262 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5263 {
5264     \bool_lazy_or:nnTF
5265     { \int_compare_p:nNn { #4 } = \c_zero_int }
5266     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c_jCol + 1 } } }
5267     \prg_return_false:
5268     \prg_return_true:
5269 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5270 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5271 {
5272     \bool_if:nTF
5273     {
5274         \int_compare_p:n { #1 <= \l_tmpa_int }
5275         &&
5276         \int_compare_p:n { \l_tmpa_int <= #3 }
5277     }
5278     \prg_return_true:
5279     \prg_return_false:
5280 }

```

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`).

```

5281 \cs_new_protected:Npn \@@_cartesian_path:n #1
5282 {
5283     \bool_lazy_and:nnT
5284     { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5285     { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5286     {

```

```

5287 \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5288 \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5289 }

```

We begin the loop over the columns.

```

5290 \clist_map_inline:Nn \l_@@_cols_tl
5291 {
5292   \tl_set:Nn \l_tmpa_tl { ##1 }
5293   \tl_if_in:NnTF \l_tmpa_tl { - }
5294     { \@@_cut_on_hyphen:w ##1 \q_stop }
5295     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5296   \bool_lazy_or:nnT
5297     { \tl_if_blank_p:V \l_tmpa_tl }
5298     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5299     { \tl_set:Nn \l_tmpa_tl { 1 } }
5300   \bool_lazy_or:nnT
5301     { \tl_if_blank_p:V \l_tmpb_tl }
5302     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5303     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5304   \int_compare:nNnT \l_tmpb_tl > \c@jCol
5305     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5306 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl

```

If we decide to provide the commands `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors` in the code-before of a `\SubMatrix`, we will have to modify the following line, by adding a kind of offset. We will have also some other lines to modify.

```

5307 \@@_qpoint:n { col - \l_tmpa_tl }
5308 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5309   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5310   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5311 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5312 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5313 \clist_map_inline:Nn \l_@@_rows_tl
5314 {
5315   \tl_set:Nn \l_tmpa_tl { #####1 }
5316   \tl_if_in:NnTF \l_tmpa_tl { - }
5317     { \@@_cut_on_hyphen:w #####1 \q_stop }
5318     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5319   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5320   \tl_if_empty:NT \l_tmpb_tl
5321     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5322   \int_compare:nNnT \l_tmpb_tl > \c@iRow
5323     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```

5324 \seq_if_in:NxF \l_@@_corners_cells_seq
5325   { \l_tmpa_tl - \l_@@_tmpc_tl }
5326   {
5327     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5328     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5329     \@@_qpoint:n { row - \l_tmpa_tl }
5330     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5331     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5332     \pgfpathrectanglecorners
5333       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5334       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5335   }
5336 }
5337 }
5338 }

```

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`).

```
5339 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }
```

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.

```
5340 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5341 {
5342   \clist_set_eq:NN \l_tmpa_clist #1
5343   \clist_clear:N #1
5344   \clist_map_inline:Nn \l_tmpa_clist
5345   {
5346     \tl_set:Nn \l_tmpa_tl { ##1 }
5347     \tl_if_in:NnTF \l_tmpa_tl { - }
5348     { \@@_cut_on_hyphen:w ##1 \q_stop }
5349     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5350     \bool_lazy_or:nnT
5351     { \tl_if_blank_p:V \l_tmpa_tl }
5352     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5353     { \tl_set:Nn \l_tmpa_tl { 1 } }
5354     \bool_lazy_or:nnT
5355     { \tl_if_blank_p:V \l_tmpb_tl }
5356     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5357     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5358     \int_compare:nNnT \l_tmpb_tl > #2
5359     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5360     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5361     { \clist_put_right:Nn #1 { #####1 } }
5362   }
5363 }
```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```
5364 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5365 {
5366   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5367   {
```

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`).

```
5368     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5369     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5370   }
5371   \ignorespaces
5372 }
```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the tabular.

```
5373 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5374 {
5375   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5376   {
5377     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5378     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5379     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5380   }
5381   \ignorespaces
5382 }
```

```

5383 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5384 {

```

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).

```

5385 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5386 {

```

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).

```

5387 \tl_gput_left:Nx \g_@@_pre_code_before_tl
5388 {
5389 \exp_not:N \columncolor [ #1 ]
5390 { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5391 }
5392 }
5393 }

```

## 22 The vertical and horizontal rules

### OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

5394 \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`.

```

5395 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5396 {
5397 \int_compare:nNnTF \l_@@_first_col_int = 0
5398 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5399 {
5400 \int_compare:nNnTF \c@jCol = 0
5401 {
5402 \int_compare:nNnF \c@iRow = { -1 }
5403 { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5404 }
5405 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5406 }
5407 }

```

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.

```

5408 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5409 {
5410 \int_compare:nNnF \c@iRow = 0
5411 { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
5412 }

```



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.

```

5413 \keys_define:nn { NiceMatrix / Rules }
5414 {
5415     position .int_set:N = \l_@@_position_int ,
5416     position .value_required:n = true ,
5417     start .int_set:N = \l_@@_start_int ,
5418     start .initial:n = 1 ,
5419     end .code:n =
5420         \bool_lazy_or:nnTF
5421             { \tl_if_empty_p:n { #1 } }
5422             { \str_if_eq_p:nn { #1 } { last } }
5423             { \int_set_eq:NN \l_@@_end_int \c@jCol }
5424             { \int_set:Nn \l_@@_end_int { #1 } }
5425 }
```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

5426 \keys_define:nn { NiceMatrix / RulesBis }
5427 {
5428     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5429     multiplicity .initial:n = 1 ,
5430     dotted .bool_set:N = \l_@@_dotted_bool ,
5431     dotted .initial:n = false ,
5432     dotted .default:n = true ,
5433     color .code:n = \@@_set_CT@arc@:n { #1 } ,
5434     color .value_required:n = true ,
5435     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5436     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`.

```

5437     tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5438     tikz .value_required:n = true ,
5439     tikz .initial:n = ,
5440     total-width .dim_set:N = \l_@@_rule_width_dim ,
5441     total-width .value_required:n = true ,
5442     width .meta:n = { total-width = #1 } ,
5443     unknown .code:n = \@@_error:n { Unknow-key-for~RulesBis }
5444 }
```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5445 \cs_new_protected:Npn \@@_vline:n #1
5446 {
```

The group is for the options.

```

5447 \group_begin:
5448 \int_zero_new:N \l_@@_end_int
5449 \int_set_eq:NN \l_@@_end_int \c@iRow
5450 \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

5451 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5452 \@@_vline_i:
5453 \group_end:
5454 }

5455 \cs_new_protected:Npn \@@_vline_i:
5456 {
5457 \int_zero_new:N \l_@@_local_start_int
5458 \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 row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

5459 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5460 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5461 \l_tmpa_tl
5462 {

```

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.

```

5463 \bool_gset_true:N \g_tmpa_bool
5464 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5465 { \@@_test_vline_in_block:nnnnn #1 }
5466 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5467 { \@@_test_vline_in_block:nnnnn #1 }
5468 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5469 { \@@_test_vline_in_stroken_block:nnnn #1 }
5470 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5471 \bool_if:NTF \g_tmpa_bool
5472 {
5473 \int_compare:nNnT \l_@@_local_start_int = 0

```

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.

```

5474 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5475 }
5476 {
5477 \int_compare:nNnT \l_@@_local_start_int > 0
5478 {
5479 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5480 \@@_vline_ii:
5481 \int_zero:N \l_@@_local_start_int
5482 }
5483 }
5484 }
5485 \int_compare:nNnT \l_@@_local_start_int > 0
5486 {
5487 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5488 \@@_vline_ii:
5489 }
5490 }

```

```

5491 \cs_new_protected:Npn \@@_test_in_corner_v:
5492 {
5493 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }

```

```

5494 {
5495   \seq_if_in:NxT
5496   \l_@@_corners_cells_seq
5497   { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5498   { \bool_set_false:N \g_tmpa_bool }
5499 }
5500 {
5501   \seq_if_in:NxT
5502   \l_@@_corners_cells_seq
5503   { \l_tmpa_tl - \l_tmpb_tl }
5504   {
5505     \int_compare:nNnTF \l_tmpb_tl = 1
5506     { \bool_set_false:N \g_tmpa_bool }
5507     {
5508       \seq_if_in:NxT
5509       \l_@@_corners_cells_seq
5510       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5511       { \bool_set_false:N \g_tmpa_bool }
5512     }
5513   }
5514 }
5515 }

5516 \cs_new_protected:Npn \@@_vline_ii:
5517 {
5518   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5519   \bool_if:NTF \l_@@_dotted_bool
5520   \@@_vline_iv:
5521   {
5522     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5523     \@@_vline_iii:
5524     \@@_vline_v:
5525   }
5526 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

5527 \cs_new_protected:Npn \@@_vline_iii:
5528 {
5529   \pgfpicture
5530   \pgfrememberpicturepositiononpagetrue
5531   \pgf@relevantforpicturesizefalse
5532   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5533   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5534   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5535   \dim_set:Nn \l_tmpb_dim
5536   {
5537     \pgf@x
5538     - 0.5 \l_@@_rule_width_dim
5539     +
5540     ( \arrayrulewidth * \l_@@_multiplicity_int
5541       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5542   }
5543   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5544   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5545   \bool_lazy_all:nT
5546   {
5547     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5548     { \cs_if_exist_p:N \CT@drsc@ }
5549     { ! \tl_if_blank_p:V \CT@drsc@ }
5550   }
5551   {
5552     \group_begin:

```

```

5553 \CT@drsc@
5554 \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5555 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5556 \dim_set:Nn \l_@@_tmpd_dim
5557 {
5558   \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5559   * ( \l_@@_multiplicity_int - 1 )
5560 }
5561 \pgfpathrectanglecorners
5562 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5563 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5564 \pgfusepath { fill }
5565 \group_end:
5566 }
5567 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5568 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5569 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5570 {
5571   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5572   \dim_sub:Nn \l_tmpb_dim \doublerulesep
5573   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5574   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5575 }
5576 \CT@arc@
5577 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5578 \pgfsetrectcap
5579 \pgfusepathqstroke
5580 \endpgfpicture
5581 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

5582 \cs_new_protected:Npn \@@_vline_iv:
5583 {
5584   \pgfpicture
5585   \pgfrememberpicturepositiononpagetrue
5586   \pgf@relevantforpicturesizefalse
5587   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5588   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5589   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5590   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5591   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5592   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5593   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5594   \CT@arc@
5595   \@@_draw_line:
5596   \endpgfpicture
5597 }

```

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`).

```

5598 \cs_new_protected:Npn \@@_vline_v:
5599 {
5600   \begin {tikzpicture }
5601   \pgfrememberpicturepositiononpagetrue
5602   \pgf@relevantforpicturesizefalse
5603   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5604   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5605   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5606   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5607   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5608   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5609   \exp_args:NV \tikzset \l_@@_tikz_rule_tl

```

```

5610 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5611 ( \l_tmpb_dim , \l_tmpa_dim ) --
5612 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5613 \end { tikzpicture }
5614 }

```

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).

```

5615 \cs_new_protected:Npn \@@_draw_vlines:
5616 {
5617   \int_step_inline:nnn
5618   {
5619     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5620     1 2
5621   }
5622   {
5623     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5624     { \int_eval:n { \c@jCol + 1 } }
5625     \c@jCol
5626   }
5627   {
5628     \tl_if_eq:NnF \l_@@_vlines_clist { all }
5629     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5630     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5631   }
5632 }

```

## 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}`.

```

5633 \cs_new_protected:Npn \@@_hline:n #1
5634 {

```

The group is for the options.

```

5635   \group_begin:
5636   \int_zero_new:N \l_@@_end_int
5637   \int_set_eq:NN \l_@@_end_int \c@jCol
5638   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5639   \@@_hline_i:
5640   \group_end:
5641 }
5642 \cs_new_protected:Npn \@@_hline_i:
5643 {
5644   \int_zero_new:N \l_@@_local_start_int
5645   \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`.

```

5646   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5647   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5648   \l_tmpb_tl
5649   {

```

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.

```

5650   \bool_gset_true:N \g_tmpa_bool
5651   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5652   { \@@_test_hline_in_block:nnnnn ##1 }
5653   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq

```

```

5654 { \@@_test_hline_in_block:nnnnn ##1 }
5655 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5656 { \@@_test_hline_in_stroken_block:nnnn ##1 }
5657 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5658 \bool_if:NTF \g_tmpa_bool
5659 {
5660   \int_compare:nNnT \l_@@_local_start_int = 0

```

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.

```

5661   { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5662 }
5663 {
5664   \int_compare:nNnT \l_@@_local_start_int > 0
5665   {
5666     \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5667     \@@_hline_ii:
5668     \int_zero:N \l_@@_local_start_int
5669   }
5670 }
5671 }
5672 \int_compare:nNnT \l_@@_local_start_int > 0
5673 {
5674   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5675   \@@_hline_ii:
5676 }
5677 }

```

```

5678 \cs_new_protected:Npn \@@_test_in_corner_h:
5679 {
5680   \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5681   {
5682     \seq_if_in:NxT
5683     \l_@@_corners_cells_seq
5684     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5685     { \bool_set_false:N \g_tmpa_bool }
5686   }
5687   {
5688     \seq_if_in:NxT
5689     \l_@@_corners_cells_seq
5690     { \l_tmpa_tl - \l_tmpb_tl }
5691     {
5692       \int_compare:nNnTF \l_tmpa_tl = 1
5693       { \bool_set_false:N \g_tmpa_bool }
5694       {
5695         \seq_if_in:NxT
5696         \l_@@_corners_cells_seq
5697         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5698         { \bool_set_false:N \g_tmpa_bool }
5699       }
5700     }
5701   }
5702 }

```

```

5703 \cs_new_protected:Npn \@@_hline_ii:
5704 {
5705   % \bool_set_false:N \l_@@_dotted_bool
5706   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5707   \bool_if:NTF \l_@@_dotted_bool
5708   \@@_hline_iv:
5709   {
5710     \tl_if_empty:NTF \l_@@_tikz_rule_tl

```

```

5711         \@@_hline_iii:
5712         \@@_hline_v:
5713     }
5714 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

5715 \cs_new_protected:Npn \@@_hline_iii:
5716 {
5717     \pgfpicture
5718     \pgfrememberpicturepositiononpagetrue
5719     \pgf@relevantforpicturesizefalse
5720     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5721     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5722     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5723     \dim_set:Nn \l_tmpb_dim
5724     {
5725         \pgf@y
5726         - 0.5 \l_@@_rule_width_dim
5727         +
5728         ( \arrayrulewidth * \l_@@_multiplicity_int
5729           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5730     }
5731     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5732     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5733     \bool_lazy_all:nT
5734     {
5735         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5736         { \cs_if_exist_p:N \CT@drsc@ }
5737         { ! \tl_if_blank_p:V \CT@drsc@ }
5738     }
5739     {
5740         \group_begin:
5741         \CT@drsc@
5742         \dim_set:Nn \l_@@_tmpd_dim
5743         {
5744             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5745             * ( \l_@@_multiplicity_int - 1 )
5746         }
5747         \pgfpathrectanglecorners
5748         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5749         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5750         \pgfusepathqfill
5751         \group_end:
5752     }
5753     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5754     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5755     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5756     {
5757         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5758         \dim_sub:Nn \l_tmpb_dim \doublerulesep
5759         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5760         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5761     }
5762     \CT@arc@
5763     \pgfsetlinewidth { 1.1 \arrayrulewidth }
5764     \pgfsetrectcap
5765     \pgfusepathqstroke
5766     \endpgfpicture
5767 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

5768 \cs_new_protected:Npn \@@_hline_iv:
5769 {
5770   \pgfpicture
5771   \pgfrememberpicturepositiononpagetrue
5772   \pgf@relevantforpicturesizefalse
5773   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5774   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5775   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
5776   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5777   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5778   \int_compare:nNnT \l_@@_local_start_int = 1
5779   {
5780     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5781     \bool_if:NT \g_@@_NiceArray_bool
5782     { \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.

```

5783   \tl_if_eq:NnF \g_@@_left_delim_tl (
5784     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
5785   )
5786   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5787   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5788   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5789   {
5790     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5791     \bool_if:NT \g_@@_NiceArray_bool
5792     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5793     \tl_if_eq:NnF \g_@@_right_delim_tl )
5794     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5795   }
5796   \CT@arc@
5797   \@@_draw_line:
5798   \endpgfpicture
5799 }

```

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`).

```

5800 \cs_new_protected:Npn \@@_hline_v:
5801 {
5802   \begin { tikzpicture }
5803   \pgfrememberpicturepositiononpagetrue
5804   \pgf@relevantforpicturesizefalse
5805   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5806   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5807   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5808   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }

```



```

5809 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5810 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5811 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5812 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5813 ( \l_tmpa_dim , \l_tmpb_dim ) --
5814 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5815 \end { tikzpicture }
5816 }

```

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)).

```

5817 \cs_new_protected:Npn \@@_draw_hlines:
5818 {
5819   \int_step_inline:nnn
5820   {
5821     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5822     1 2
5823   }
5824   {
5825     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5826     { \int_eval:n { \c@iRow + 1 } }
5827     \c@iRow
5828   }
5829   {
5830     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5831     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5832     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5833   }
5834 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5835 \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.

```

5836 \cs_set:Npn \@@_Hline_i:n #1
5837 {
5838   \peek_remove_spaces:n
5839   {
5840     \peek_meaning:NTF \Hline
5841     { \@@_Hline_ii:nn { #1 + 1 } }
5842     { \@@_Hline_iii:n { #1 } }
5843   }
5844 }
5845 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5846 \cs_set:Npn \@@_Hline_iii:n #1
5847 {
5848   \peek_meaning:NTF [
5849   { \@@_Hline_iv:nw { #1 } }
5850   { \@@_Hline_iv:nw { #1 } [ ] }
5851 }
5852 \cs_set:Npn \@@_Hline_iv:nw #1 [ #2 ]
5853 {
5854   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
5855   \skip_vertical:n { \l_@@_rule_width_dim }
5856   \tl_gput_right:Nx \g_@@_pre_code_after_tl
5857   {
5858     \@@_hline:n
5859     {
5860       multiplicity = #1 ,
5861       position = \int_eval:n { \c@iRow + 1 } ,

```

```

5862         total-width = \dim_use:N \l_@@_rule_width_dim ,
5863         #2
5864     }
5865 }
5866 \egroup
5867 }

```

### 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.

Among the keys available in that list, there is the key `letter` to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix / ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

5868 \keys_define:nn { NiceMatrix / ColumnTypes } { }

```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```

5869 \cs_new_protected:Npn \@@_custom_line:n #1
5870 {
5871     \str_clear_new:N \l_@@_command_str
5872     \str_clear_new:N \l_@@_ccommand_str
5873     \str_clear_new:N \l_@@_letter_str
5874     \keys_set_known:nn { 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.

```

5875     \bool_lazy_all:nTF
5876     {
5877         { \str_if_empty_p:N \l_@@_letter_str }
5878         { \str_if_empty_p:N \l_@@_command_str }
5879         { \str_if_empty_p:N \l_@@_ccommand_str }
5880     }
5881     { \@@_error:n { No~letter~and~no~command } }
5882     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
5883 }

5884 \keys_define:nn { NiceMatrix / custom-line }
5885 {
5886     letter .str_set:N = \l_@@_letter_str ,
5887     letter .value_required:n = true ,
5888     command .str_set:N = \l_@@_command_str ,
5889     command .value_required:n = true ,
5890     ccommand .str_set:N = \l_@@_ccommand_str ,
5891     ccommand .value_required:n = true ,
5892 }

```

```

5893 \cs_new_protected:Npn \@@_custom_line_i:n #1
5894 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

5895     \bool_set_false:N \l_@@_tikz_rule_bool
5896     \bool_set_false:N \l_@@_dotted_rule_bool
5897     \bool_set_false:N \l_@@_color_bool

```

```

5898 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
5899 \bool_if:NT \l_@@_tikz_rule_bool
5900 {
5901   \IfPackageLoadedTF { tikz }
5902   { }
5903   { \@@_error:n { tikz~in~custom~line~without~tikz } }
5904   \bool_if:NT \l_@@_color_bool
5905   { \@@_error:n { color~in~custom~line~with~tikz } }
5906 }
5907 \bool_if:nT
5908 {
5909   \int_compare_p:nNn \l_@@_multiplicity_int > 1
5910   && \l_@@_dotted_rule_bool
5911 }
5912 { \@@_error:n { key~multiplicity~with~dotted } }
5913 \str_if_empty:NF \l_@@_letter_str
5914 {
5915   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
5916   { \@@_error:n { Several~letters } }
5917   {
5918     \exp_args:NnV \tl_if_in:NnTF
5919     \c_@@_forbidden_letters_str \l_@@_letter_str
5920     { \@@_error:n { Forbidden~letter } }
5921     {

```

The final user can, locally, redefine a letter of column type. That's compatible with the use of `\keys_define:nn`: the definition is local and may overwrite a previous definition.

```

5922 \keys_define:nx { NiceMatrix / ColumnTypes }
5923 {
5924   \l_@@_letter_str .code:n =
5925   { \@@_v_custom_line:n { \exp_not:n { #1 } } }
5926 }
5927 }
5928 }
5929 }
5930 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
5931 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
5932 }
5933 \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.

```

5934 \keys_define:nn { NiceMatrix / custom-line-bis }
5935 {
5936   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5937   multiplicity .initial:n = 1 ,
5938   multiplicity .value_required:n = true ,
5939   color .code:n = \bool_set_true:N \l_@@_color_bool ,
5940   color .value_required:n = true ,
5941   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5942   tikz .value_required:n = true ,
5943   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5944   dotted .value_forbidden:n = true ,
5945   total-width .code:n = { } ,
5946   total-width .value_required:n = true ,
5947   width .code:n = { } ,
5948   width .value_required:n = true ,
5949   sep-color .code:n = { } ,
5950   sep-color .value_required:n = true ,
5951   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
5952 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

5953 \bool_new:N \l_@@_dotted_rule_bool
5954 \bool_new:N \l_@@_tikz_rule_bool
5955 \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`.

```

5956 \keys_define:nn { NiceMatrix / custom-line-width }
5957 {
5958   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5959   multiplicity .initial:n = 1 ,
5960   multiplicity .value_required:n = true ,
5961   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5962   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5963                       \bool_set_true:N \l_@@_total_width_bool ,
5964   total-width .value_required:n = true ,
5965   width .meta:n = { total-width = #1 } ,
5966   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5967 }

```

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`).

```

5968 \cs_new_protected:Npn \@@_h_custom_line:n #1
5969 {

```

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`.

```

5970   \cs_set:cpn { nicematrix - \l_@@_command_str }
5971   {
5972     \noalign
5973     {
5974       \@@_compute_rule_width:n { #1 }
5975       \skip_vertical:n { \l_@@_rule_width_dim }
5976       \tl_gput_right:Nx \g_@@_pre_code_after_tl
5977       {
5978         \@@_hline:n
5979         {
5980           #1 ,
5981           position = \int_eval:n { \c@iRow + 1 } ,
5982           total-width = \dim_use:N \l_@@_rule_width_dim
5983         }
5984       }
5985     }
5986   }
5987   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
5988 }
5989 \cs_generate_variant:Nn \@@_h_custom_line:nn { n V }

```

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`).

```

5990 \cs_new_protected:Npn \@@_c_custom_line:n #1
5991 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

5992   \exp_args:Nc \NewExpandableDocumentCommand
5993   { nicematrix - \l_@@_ccommand_str }
5994   { 0 { } m }

```

```

5995 {
5996   \noalign
5997   {
5998     \@@_compute_rule_width:n { #1 , ##1 }
5999     \skip_vertical:n { \l_@@_rule_width_dim }
6000     \clist_map_inline:nn
6001       { ##2 }
6002       { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6003   }
6004 }
6005 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6006 }

```

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*.

```

6007 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6008 {
6009   \str_if_in:nnTF { #2 } { - }
6010   { \@@_cut_on_hyphen:w #2 \q_stop }
6011   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6012   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6013   {
6014     \@@_hline:n
6015     {
6016       #1 ,
6017       start = \l_tmpa_tl ,
6018       end = \l_tmpb_tl ,
6019       position = \int_eval:n { \c@iRow + 1 } ,
6020       total-width = \dim_use:N \l_@@_rule_width_dim
6021     }
6022   }
6023 }
6024 \cs_generate_variant:Nn \@@_c_custom_line:nn { n V }
6025 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6026 {
6027   \bool_set_false:N \l_@@_tikz_rule_bool
6028   \bool_set_false:N \l_@@_total_width_bool
6029   \bool_set_false:N \l_@@_dotted_rule_bool
6030   \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6031   \bool_if:NF \l_@@_total_width_bool
6032   {
6033     \bool_if:NTF \l_@@_dotted_rule_bool
6034     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6035     {
6036       \bool_if:NF \l_@@_tikz_rule_bool
6037       {
6038         \dim_set:Nn \l_@@_rule_width_dim
6039         {
6040           \arrayrulewidth * \l_@@_multiplicity_int
6041           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6042         }
6043       }
6044     }
6045   }
6046 }
6047 \cs_new_protected:Npn \@@_v_custom_line:n #1
6048 {
6049   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6050 \tl_gput_right:Nx \g_@@_preamble_tl
6051 { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6052 \tl_gput_right:Nx \g_@@_pre_code_after_tl

```

```

6053     {
6054         \@@_vline:n
6055         {
6056             #1 ,
6057             position = \int_eval:n { \c@jCol + 1 } ,
6058             total-width = \dim_use:N \l_@@_rule_width_dim
6059         }
6060     }
6061 }
6062 \@@_custom_line:n
6063 { 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.

```

6064 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6065 {
6066     \bool_lazy_all:nT
6067     {
6068         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6069         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6070         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6071         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6072     }
6073     { \bool_gset_false:N \g_tmpa_bool }
6074 }

```

The same for vertical rules.

```

6075 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6076 {
6077     \bool_lazy_all:nT
6078     {
6079         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6080         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6081         { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6082         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6083     }
6084     { \bool_gset_false:N \g_tmpa_bool }
6085 }
6086 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6087 {
6088     \bool_lazy_all:nT
6089     {
6090         {
6091             ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6092             || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6093         }
6094         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6095         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6096     }
6097     { \bool_gset_false:N \g_tmpa_bool }
6098 }
6099 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6100 {
6101     \bool_lazy_all:nT
6102     {
6103         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6104         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6105         {

```

```

6106         ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6107         || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6108     }
6109 }
6110 { \bool_gset_false:N \g_tpa_bool }
6111 }

```

## 23 The key corners

When the key `corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```

6112 \cs_new_protected:Npn \@@_compute_corners:
6113 {

```

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.

```

6114     \seq_clear_new:N \l_@@_corners_cells_seq
6115     \clist_map_inline:Nn \l_@@_corners_clist
6116     {
6117         \str_case:nnF { ##1 }
6118         {
6119             { NW }
6120             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6121             { NE }
6122             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6123             { SW }
6124             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6125             { SE }
6126             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6127         }
6128         { \@@_error:nn { bad~corner } { ##1 } }
6129     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6130     \seq_if_empty:NF \l_@@_corners_cells_seq
6131     {

```

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.

```

6132         \tl_gput_right:Nx \g_@@_aux_tl
6133         {
6134             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6135             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6136         }
6137     }
6138 }

```

“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.

```

6139 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6140 {

```

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.

```

6141 \bool_set_false:N \l_tmpa_bool
6142 \int_zero_new:N \l_@@_last_empty_row_int
6143 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6144 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6145 {
6146   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6147   \bool_lazy_or:nnTF
6148   {
6149     \cs_if_exist_p:c
6150     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6151   }
6152   \l_tmpb_bool
6153   { \bool_set_true:N \l_tmpa_bool }
6154   {
6155     \bool_if:NF \l_tmpa_bool
6156     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6157   }
6158 }

```

Now, you determine the last empty cell in the row of number 1.

```

6159 \bool_set_false:N \l_tmpa_bool
6160 \int_zero_new:N \l_@@_last_empty_column_int
6161 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6162 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6163 {
6164   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6165   \bool_lazy_or:nnTF
6166   \l_tmpb_bool
6167   {
6168     \cs_if_exist_p:c
6169     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6170   }
6171   { \bool_set_true:N \l_tmpa_bool }
6172   {
6173     \bool_if:NF \l_tmpa_bool
6174     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6175   }
6176 }

```

Now, we loop over the rows.

```

6177 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6178 {

```

We treat the row number `##1` with another loop.

```

6179 \bool_set_false:N \l_tmpa_bool
6180 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6181 {
6182   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6183   \bool_lazy_or:nnTF
6184   \l_tmpb_bool
6185   {
6186     \cs_if_exist_p:c
6187     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6188   }
6189   { \bool_set_true:N \l_tmpa_bool }
6190   {
6191     \bool_if:NF \l_tmpa_bool

```



```

6192         {
6193             \int_set:Nn \l_@@_last_empty_column_int { ###1 }
6194             \seq_put_right:Nn
6195                 \l_@@_corners_cells_seq
6196                 { ##1 - ###1 }
6197         }
6198     }
6199 }
6200 }
6201 }

```

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`).

```

6202 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6203 {
6204     \int_set:Nn \l_tmpa_int { #1 }
6205     \int_set:Nn \l_tmpb_int { #2 }
6206     \bool_set_false:N \l_tmpb_bool
6207     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6208         { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6209 }
6210 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6211 {
6212     \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6213     {
6214         \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6215         {
6216             \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6217             {
6218                 \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6219                 { \bool_set_true:N \l_tmpb_bool }
6220             }
6221         }
6222     }
6223 }

```

## 24 The environment `{NiceMatrixBlock}`

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

6224 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6225 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6226 {
6227     auto-columns-width .code:n =
6228     {
6229         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6230         \dim_gzero_new:N \g_@@_max_cell_width_dim
6231         \bool_set_true:N \l_@@_auto_columns_width_bool
6232     }
6233 }

```

```

6234 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6235 {
6236   \int_gincr:N \g_@@_NiceMatrixBlock_int
6237   \dim_zero:N \l_@@_columns_width_dim
6238   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6239   \bool_if:NT \l_@@_block_auto_columns_width_bool
6240   {
6241     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6242     {
6243       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
6244         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6245     }
6246   }
6247 }

```

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).

```

6248 {
6249   \bool_if:NT \l_@@_block_auto_columns_width_bool
6250   {
6251     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6252     \iow_shipout:Nx \@mainaux
6253     {
6254       \cs_gset:cpn
6255         { @@_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.

```

6256       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6257     }
6258     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6259   }
6260 }

```

## 25 The extra nodes

First, two variants of the functions \dim\_min:nn and \dim\_max:nn.

```

6261 \cs_generate_variant:Nn \dim_min:nn { v n }
6262 \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).

```

6263 \cs_new_protected:Npn \@@_create_extra_nodes:
6264 {
6265   \bool_if:nTF \l_@@_medium_nodes_bool
6266   {
6267     \bool_if:nTF \l_@@_large_nodes_bool
6268     \@@_create_medium_and_large_nodes:
6269     \@@_create_medium_nodes:
6270   }
6271   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6272 }

```

We have three macros of creation of nodes: \@@\_create\_medium\_nodes:, \@@\_create\_large\_nodes: and \@@\_create\_medium\_and\_large\_nodes:.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command `\@@_computations_for_medium_nodes:` to do these computations.

The command `\@@_computations_for_medium_nodes:` must be used in a `{pgfpicture}`.

For each row  $i$ , we compute two dimensions `l_@@_row_i_min_dim` and `l_@@_row_i_max_dim`. The dimension `l_@@_row_i_min_dim` is the minimal  $y$ -value of all the cells of the row  $i$ . The dimension `l_@@_row_i_max_dim` is the maximal  $y$ -value of all the cells of the row  $i$ .

Similarly, for each column  $j$ , we compute two dimensions `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. The dimension `l_@@_column_j_min_dim` is the minimal  $x$ -value of all the cells of the column  $j$ . The dimension `l_@@_column_j_max_dim` is the maximal  $x$ -value of all the cells of the column  $j$ .

Since these dimensions will be computed as maximum or minimum, we initialize them to `\c_max_dim` or `-\c_max_dim`.

```

6273 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6274 {
6275   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6276   {
6277     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6278     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6279     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6280     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6281   }
6282   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6283   {
6284     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6285     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6286     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6287     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6288   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6289   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6290   {
6291     \int_step_variable:nnNn
6292     \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.

```

6293     {
6294       \cs_if_exist:cT
6295       { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor `south west` of the (normal) node of the cell  $(i-j)$ . They will be stored in `\pgf@x` and `\pgf@y`.

```

6296     {
6297       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6298       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6299       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6300       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6301       {
6302         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6303         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6304       }

```

We retrieve the coordinates of the anchor `north east` of the (normal) node of the cell  $(i-j)$ . They will be stored in `\pgf@x` and `\pgf@y`.

```

6305       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6306       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6307       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6308       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6309       {
6310         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }

```

```

6311         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6312     }
6313 }
6314 }
6315 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6316 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6317 {
6318     \dim_compare:nNnT
6319     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6320     {
6321         \@@_qpoint:n { row - \@@_i: - base }
6322         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6323         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6324     }
6325 }
6326 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6327 {
6328     \dim_compare:nNnT
6329     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6330     {
6331         \@@_qpoint:n { col - \@@_j: }
6332         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6333         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6334     }
6335 }
6336 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6337 \cs_new_protected:Npn \@@_create_medium_nodes:
6338 {
6339     \pgfpicture
6340     \pgfrememberpicturepositiononpagetrue
6341     \pgf@relevantforpicturesizefalse
6342     \@@_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”.

```

6343     \tl_set:Nn \l_@@_suffix_tl { -medium }
6344     \@@_create_nodes:
6345     \endpgfpicture
6346 }

```

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:`.

```

6347 \cs_new_protected:Npn \@@_create_large_nodes:
6348 {
6349     \pgfpicture
6350     \pgfrememberpicturepositiononpagetrue
6351     \pgf@relevantforpicturesizefalse
6352     \@@_computations_for_medium_nodes:
6353     \@@_computations_for_large_nodes:
6354     \tl_set:Nn \l_@@_suffix_tl { - large }
6355     \@@_create_nodes:

```

---

<sup>14</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6356 \endpgfpicture
6357 }
6358 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6359 {
6360 \pgfpicture
6361 \pgfrememberpicturepositiononpagetrue
6362 \pgf@relevantforpicturesizefalse
6363 \@@_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”.

```

6364 \tl_set:Nn \l_@@_suffix_tl { - medium }
6365 \@@_create_nodes:
6366 \@@_computations_for_large_nodes:
6367 \tl_set:Nn \l_@@_suffix_tl { - large }
6368 \@@_create_nodes:
6369 \endpgfpicture
6370 }

```

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.

```

6371 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6372 {
6373 \int_set:Nn \l_@@_first_row_int 1
6374 \int_set:Nn \l_@@_first_col_int 1

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

6375 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6376 {
6377 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6378 {
6379 (
6380 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6381 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6382 )
6383 / 2
6384 }
6385 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6386 { l_@@_row _ \@@_i: _ min_dim }
6387 }
6388 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6389 {
6390 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6391 {
6392 (
6393 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6394 \dim_use:c
6395 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6396 )
6397 / 2
6398 }
6399 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6400 { l_@@_column _ \@@_j: _ max _ dim }
6401 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6402 \dim_sub:cn
6403 { l_@@_column _ 1 _ min _ dim }
6404 \l_@@_left_margin_dim
6405 \dim_add:cn
6406 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6407 \l_@@_right_margin_dim
6408 }

```

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`).

```

6409 \cs_new_protected:Npn \@@_create_nodes:
6410 {
6411   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6412   {
6413     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6414     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6415       \@@_pgf_rect_node:nnnnn
6416       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6417       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6418       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6419       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6420       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6421       \str_if_empty:NF \l_@@_name_str
6422       {
6423         \pgfnodealias
6424         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6425         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6426       }
6427     }
6428   }

```

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  $n$ .

```

6429   \cs_if_exist_use:NF
6430   \seq_map_pairwise_function:NNN
6431   \seq_mapthread_function:NNN
6432   \g_@@_multicolumn_cells_seq
6433   \g_@@_multicolumn_sizes_seq
6434   \@@_node_for_multicolumn:nn
6435 }

```

```

6436 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6437 {
6438   \cs_set_nopar:Npn \@@_i: { #1 }
6439   \cs_set_nopar:Npn \@@_j: { #2 }
6440 }

```

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”).

```

6441 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6442 {
6443   \@@_extract_coords_values: #1 \q_stop
6444   \@@_pgf_rect_node:nnnnn
6445   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6446   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6447   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6448   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
6449   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6450   \str_if_empty:NF \l_@@_name_str
6451   {
6452     \pgfnodealias
6453     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6454     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}

```

```

6455     }
6456 }

```

## 26 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.

```

6457 \keys_define:nn { NiceMatrix / Block / FirstPass }
6458 {
6459   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6460   l .value_forbidden:n = true ,
6461   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6462   r .value_forbidden:n = true ,
6463   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6464   c .value_forbidden:n = true ,
6465   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6466   L .value_forbidden:n = true ,
6467   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6468   R .value_forbidden:n = true ,
6469   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6470   C .value_forbidden:n = true ,
6471   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6472   t .value_forbidden:n = true ,
6473   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6474   b .value_forbidden:n = true ,
6475   color .tl_set:N = \l_@@_color_tl ,
6476   color .value_required:n = true ,
6477   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6478   respect-arraystretch .default:n = true ,
6479 }

```

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.

```

6480 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
6481 {

```

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).

```

6482   \peek_remove_spaces:n
6483   {
6484     \tl_if_blank:nTF { #2 }
6485     { \@@_Block_i 1-1 \q_stop }
6486     {
6487       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6488       \@@_Block_i_czech \@@_Block_i
6489       #2 \q_stop
6490     }
6491     { #1 } { #3 } { #4 }
6492   }
6493 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

6494 \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`.

```

6495 {
6496   \char_set_catcode_active:N -
6497   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6498 }

```

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 the beginning of the small array of the block and `#5` is the label of the block.

```

6499 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6500 {

```

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).

```

6501   \bool_lazy_or:nnTF
6502     { \tl_if_blank_p:n { #1 } }
6503     { \str_if_eq_p:nn { #1 } { * } }
6504     { \int_set:Nn \l_tmpa_int { 100 } }
6505     { \int_set:Nn \l_tmpa_int { #1 } }
6506   \bool_lazy_or:nnTF
6507     { \tl_if_blank_p:n { #2 } }
6508     { \str_if_eq_p:nn { #2 } { * } }
6509     { \int_set:Nn \l_tmpb_int { 100 } }
6510     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6511   \int_compare:nNnTF \l_tmpb_int = 1
6512   {
6513     \str_if_empty:NTF \l_@@_hpos_cell_str
6514     { \str_set:Nn \l_@@_hpos_block_str c }
6515     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6516   }
6517   { \str_set:Nn \l_@@_hpos_block_str c }

```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```

6518   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6519   \tl_set:Nx \l_tmpa_tl
6520   {
6521     { \int_use:N \c@iRow }
6522     { \int_use:N \c@jCol }
6523     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6524     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6525   }

```

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).

```

6526   \bool_if:nTF
6527   {
6528     (
6529       \int_compare_p:nNn { \l_tmpa_int } = 1
6530       ||

```



```

6531     \int_compare:nNn { \l_tmpb_int } = 1
6532   )
6533   && ! \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.

```

6534     && ! \l_@@_X_column_bool
6535   }
6536   { \exp_args:Nxx \@@_Block_iv:nnnnn }
6537   { \exp_args:Nxx \@@_Block_v:nnnnn }
6538   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6539 }

```

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).

```

6540 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6541 {
6542   \int_gincr:N \g_@@_block_box_int
6543   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6544   {
6545     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6546     {
6547       \@@_actually_diagbox:nnnnnn
6548       { \int_use:N \c@iRow }
6549       { \int_use:N \c@jCol }
6550       { \int_eval:n { \c@iRow + #1 - 1 } }
6551       { \int_eval:n { \c@jCol + #2 - 1 } }
6552       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6553     }
6554   }
6555   \box_gclear_new:c
6556   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
6557   \hbox_gset:cn
6558   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
6559   {

```

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}).

```

6560     \tl_if_empty:NTF \l_@@_color_tl
6561     { \int_compare:nNnT { #2 } = 1 \set@color }
6562     { \@@_color:V \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.

```

6563     \int_compare:nNnT { #1 } = 1
6564     {
6565       \int_compare:nNnTF \c@iRow = 0
6566       \l_@@_code_for_first_row_tl
6567       {
6568         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6569         \l_@@_code_for_last_row_tl
6570       }
6571       \g_@@_row_style_tl
6572     }

```

```

6573 \group_begin:
6574 \bool_if:NF \l_@@_respect_arraystretch_bool
6575 { \cs_set:Npn \arraystretch { 1 } }
6576 \dim_zero:N \extrarowheight
6577 #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).

```

6578 \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
6579 \bool_if:NTF \l_@@_NiceTabular_bool
6580 {
6581   \bool_lazy_all:nTF
6582   {
6583     { \int_compare_p:nNn { #2 } = 1 }
6584     { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6585     { ! \g_@@_rotate_bool }
6586   }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

6587 {
6588   \use:x
6589   {
6590     \exp_not:N \begin { minipage }%
6591     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6592     { \l_@@_col_width_dim }
6593     \str_case:Vn \l_@@_hpos_block_str
6594     {
6595       c \centering
6596       r \raggedleft
6597       l \raggedright
6598     }
6599   }
6600   #5
6601   \end { minipage }
6602 }
6603 {
6604   \use:x
6605   {
6606     \exp_not:N \begin { tabular }%
6607     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6608     { @ { } \l_@@_hpos_block_str @ { } }
6609   }
6610   #5
6611   \end { tabular }
6612 }
6613 {
6614   \c_math_toggle_token
6615   \use:x
6616   {
6617     \exp_not:N \begin { array }%
6618     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6619     { @ { } \l_@@_hpos_block_str @ { } }
6620   }
6621   #5
6622   \end { array }
6623   \c_math_toggle_token
6624 }
6625 \group_end:
6626 }
6627 \bool_if:NT \g_@@_rotate_bool
6628

```

```

6629 {
6630   \box_grotate:cn
6631   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6632   { 90 }
6633   \bool_gset_false:N \g_@@_rotate_bool
6634 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

6635 \int_compare:nNnT { #2 } = 1
6636 {
6637   \dim_gset:Nn \g_@@_blocks_wd_dim
6638   {
6639     \dim_max:nn
6640     \g_@@_blocks_wd_dim
6641     {
6642       \box_wd:c
6643       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6644     }
6645   }
6646 }

```

If we are in a mono-row block, we take into account the height and the depth of that block for the height and the depth of the row.

```

6647 \int_compare:nNnT { #1 } = 1
6648 {
6649   \dim_gset:Nn \g_@@_blocks_ht_dim
6650   {
6651     \dim_max:nn
6652     \g_@@_blocks_ht_dim
6653     {
6654       \box_ht:c
6655       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6656     }
6657   }
6658   \dim_gset:Nn \g_@@_blocks_dp_dim
6659   {
6660     \dim_max:nn
6661     \g_@@_blocks_dp_dim
6662     {
6663       \box_dp:c
6664       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6665     }
6666   }
6667 }
6668 \seq_gput_right:Nx \g_@@_blocks_seq
6669 {
6670   \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.

```

6671 { \exp_not:n { #3 } , \l_@@_hpos_block_str }
6672 {
6673   \box_use_drop:c
6674   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6675 }
6676 }
6677 }

```

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.

```

6678 \cs_new_protected:Npn \l_@@_Block_v:nnnnn #1 #2 #3 #4 #5
6679 {
6680   \seq_gput_right:Nx \g_@@_blocks_seq
6681   {
6682     \l_tmpa_tl
6683     { \exp_not:n { #3 } }
6684     {
6685       \bool_if:NTF \l_@@_NiceTabular_bool
6686       {
6687         \group_begin:
6688         \bool_if:NF \l_@@_respect_arraystretch_bool
6689         { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6690         \exp_not:n
6691         {
6692           \dim_zero:N \extrarowheight
6693           #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).

```

6694       \bool_if:NT \g_@@_rotate_bool
6695       { \str_set:Nn \l_@@_hpos_block_str c }
6696       \use:x
6697       {
6698         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
6699         { @ { } \l_@@_hpos_block_str @ { } }
6700       }
6701       #5
6702       \end { tabular }
6703     }
6704   \group_end:
6705 }
6706 {
6707   \group_begin:
6708   \bool_if:NF \l_@@_respect_arraystretch_bool
6709   { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6710   \exp_not:n
6711   {
6712     \dim_zero:N \extrarowheight
6713     #4
6714     \bool_if:NT \g_@@_rotate_bool
6715     { \str_set:Nn \l_@@_hpos_block_str c }
6716     \c_math_toggle_token
6717     \use:x
6718     {
6719       \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
6720       { @ { } \l_@@_hpos_block_str @ { } }
6721     }
6722     #5
6723     \end { array }
6724     \c_math_toggle_token
6725   }
6726   \group_end:
6727 }
6728 }
6729 }
6730 }

```

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).

```

6731 \keys_define:nn { NiceMatrix / Block / SecondPass }
6732 {
6733   tikz .code:n =
6734     \IfPackageLoadedTF { tikz }
6735       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6736       { \@@_error:n { tikz~key~without~tikz } } ,
6737   tikz .value_required:n = true ,
6738   fill .code:n =
6739     \tl_set_rescan:Nnn
6740       \l_@@_fill_tl
6741       { \char_set_catcode_other:N ! }
6742       { #1 } ,
6743   fill .value_required:n = true ,
6744   draw .code:n =
6745     \tl_set_rescan:Nnn
6746       \l_@@_draw_tl
6747       { \char_set_catcode_other:N ! }
6748       { #1 } ,
6749   draw .default:n = default ,
6750   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6751   rounded-corners .default:n = 4 pt ,
6752   color .code:n =
6753     \@@_color:n { #1 }
6754     \tl_set_rescan:Nnn
6755       \l_@@_draw_tl
6756       { \char_set_catcode_other:N ! }
6757       { #1 } ,
6758   color .value_required:n = true ,
6759   borders .clist_set:N = \l_@@_borders_clist ,
6760   borders .value_required:n = true ,
6761   hvlines .meta:n = { vlines , hlines } ,
6762   vlines .bool_set:N = \l_@@_vlines_block_bool ,
6763   vlines .default:n = true ,
6764   hlines .bool_set:N = \l_@@_hlines_block_bool ,
6765   hlines .default:n = true ,
6766   line-width .dim_set:N = \l_@@_line_width_dim ,
6767   line-width .value_required:n = true ,
6768   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6769   l .value_forbidden:n = true ,
6770   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6771   r .value_forbidden:n = true ,
6772   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6773   c .value_forbidden:n = true ,
6774   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6775     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6776   L .value_forbidden:n = true ,
6777   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6778     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6779   R .value_forbidden:n = true ,
6780   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6781     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6782   C .value_forbidden:n = true ,
6783   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6784   t .value_forbidden:n = true ,
6785   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6786   T .value_forbidden:n = true ,
6787   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6788   b .value_forbidden:n = true ,
6789   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6790   B .value_forbidden:n = true ,
6791   v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
6792   v-center .value_forbidden:n = true ,
6793   name .tl_set:N = \l_@@_block_name_str ,

```

```

6794   name .value_required:n = true ,
6795   name .initial:n = ,
6796   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6797   respect-arraystretch .default:n = true ,
6798   transparent .bool_set:N = \l_@@_transparent_bool ,
6799   transparent .default:n = true ,
6800   transparent .initial:n = false ,
6801   unknown .code:n = \@@_error:n { Unknown-key-for-Block }
6802 }

```

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.

```

6803 \cs_new_protected:Npn \@@_draw_blocks:
6804 {
6805   \cs_set_eq:NN \ialign \@@_old_ialign:
6806   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
6807 }
6808 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6809 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6810   \int_zero_new:N \l_@@_last_row_int
6811   \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.

```

6812   \int_compare:nNnTF { #3 } > { 99 }
6813   { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
6814     { \int_set:Nn \l_@@_last_row_int { #3 } }
6815   \int_compare:nNnTF { #4 } > { 99 }
6816   { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
6817     { \int_set:Nn \l_@@_last_col_int { #4 } }
6818   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6819   {
6820     \int_compare:nTF
6821     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6822     {
6823       \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
6824       \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
6825       \@@_msg_redirect_name:nn { columns-not-used } { none }
6826     }
6827     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6828   }
6829   {
6830     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6831     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6832     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6833   }
6834 }

```

**#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

```

6835 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6836 {

```

The group is for the keys.

```

6837 \group_begin:
6838 \int_compare:nNnT { #1 } = { #3 }
6839 { \str_set:Nn \l_@@_vpos_of_block_str { t } }
6840 \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
6841 \bool_if:NT \l_@@_vlines_block_bool
6842 {
6843   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6844   {
6845     \@@_vlines_block:nnn
6846     { \exp_not:n { #5 } }
6847     { #1 - #2 }
6848     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6849   }
6850 }
6851 \bool_if:NT \l_@@_hlines_block_bool
6852 {
6853   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6854   {
6855     \@@_hlines_block:nnn
6856     { \exp_not:n { #5 } }
6857     { #1 - #2 }
6858     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6859   }
6860 }
6861 \bool_if:nF
6862 {
6863   \l_@@_transparent_bool
6864   || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
6865 }
6866 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

6867 \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6868 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6869 }

6870 \bool_lazy_and:nnT
6871 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
6872 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
6873 { \@@_error:n { hlines-with-color } }

6874 \tl_if_empty:NF \l_@@_draw_tl
6875 {
6876   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6877   {
6878     \@@_stroke_block:nnn
6879     { \exp_not:n { #5 } }
6880     { #1 - #2 }
6881     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6882   }
6883   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6884   { { #1 } { #2 } { #3 } { #4 } }
6885 }

6886 \clist_if_empty:NF \l_@@_borders_clist
6887 {
6888   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6889   {
6890     \@@_stroke_borders_block:nnn
6891     { \exp_not:n { #5 } }

```

```

6892         { #1 - #2 }
6893         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6894     }
6895 }
6896 \tl_if_empty:NF \l_@@_fill_tl
6897 {
6898     \tl_gput_right:Nx \g_@@_pre_code_before_tl
6899     {
6900         \exp_not:N \roundedrectanglecolor
6901         \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
6902             { \l_@@_fill_tl }
6903             { { \l_@@_fill_tl } }
6904             { #1 - #2 }
6905             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6906             { \dim_use:N \l_@@_rounded_corners_dim }
6907         ]
6908     }
6909 \seq_if_empty:NF \l_@@_tikz_seq
6910 {
6911     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6912     {
6913         \@@_block_tikz:nnnnn
6914         { #1 }
6915         { #2 }
6916         { \int_use:N \l_@@_last_row_int }
6917         { \int_use:N \l_@@_last_col_int }
6918         { \seq_use:Nn \l_@@_tikz_seq { , } }
6919     }
6920 }
6921 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6922 {
6923     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6924     {
6925         \@@_actually_diagbox:nnnnnn
6926         { #1 }
6927         { #2 }
6928         { \int_use:N \l_@@_last_row_int }
6929         { \int_use:N \l_@@_last_col_int }
6930         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6931     }
6932 }
6933 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6934 \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.

```

6935 \pgfpicture
6936 \pgfrememberpicturepositiononpagetrue
6937 \pgf@relevantforpicturesizefalse
6938 \@@_qpoint:n { row - #1 }
6939 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6940 \@@_qpoint:n { col - #2 }
6941 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6942 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6943 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6944 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6945 \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.

```

6946 \@@_pgf_rect_node:nnnnn
6947 { \@@_env: - #1 - #2 - block }
6948 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6949 \str_if_empty:NF \l_@@_block_name_str
6950 {
6951 \pgfnodealias
6952 { \@@_env: - \l_@@_block_name_str }
6953 { \@@_env: - #1 - #2 - block }
6954 \str_if_empty:NF \l_@@_name_str
6955 {
6956 \pgfnodealias
6957 { \l_@@_name_str - \l_@@_block_name_str }
6958 { \@@_env: - #1 - #2 - block }
6959 }
6960 }

```

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.

```

6961 \bool_if:NF \l_@@_hpos_of_block_cap_bool
6962 {
6963 \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.

```

6964 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6965 {

```

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.

```

6966 \cs_if_exist:cT
6967 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6968 {
6969 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6970 {
6971 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6972 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6973 }
6974 }
6975 }

```

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.

```

6976     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6977     {
6978         \@@_qpoint:n { col - #2 }
6979         \dim_set_eq:NN \l_tmpb_dim \pgf@x
6980     }
6981     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6982     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6983     {
6984         \cs_if_exist:cT
6985         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6986         {
6987             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6988             {
6989                 \pgfpointanchor
6990                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6991                 { east }
6992                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
6993             }
6994         }
6995     }
6996     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
6997     {
6998         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6999         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7000     }
7001     \@@_pgf_rect_node:nnnnn
7002     { \@@_env: - #1 - #2 - block - short }
7003     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7004 }

```

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.

```

7005     \bool_if:NT \l_@@_medium_nodes_bool
7006     {
7007         \@@_pgf_rect_node:nnn
7008         { \@@_env: - #1 - #2 - block - medium }
7009         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7010         {
7011             \pgfpointanchor
7012             { \@@_env:
7013                 - \int_use:N \l_@@_last_row_int
7014                 - \int_use:N \l_@@_last_col_int - medium
7015             }
7016             { south-east }
7017         }
7018     }

```

Now, we will put the label of the block.

```

7019     \bool_lazy_any:nTF
7020     {
7021         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7022         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7023         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7024     }
7025     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7026     \int_compare:nNnT { #2 } = 0
7027     { \str_set:Nn \l_@@_hpos_block_str r }
7028     \bool_if:nT \g_@@_last_col_found_bool

```

```

7029     {
7030         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7031         { \str_set:Nn \l_@@_hpos_block_str 1 }
7032     }
7033 \tl_set:Nx \l_tmpa_tl
7034 {
7035     \str_case:Vn \l_@@_vpos_of_block_str
7036     {
7037         c {
7038             \str_case:Vn \l_@@_hpos_block_str
7039             {
7040                 c { center }
7041                 l { west }
7042                 r { east }
7043             }
7044
7045         }
7046         T {
7047             \str_case:Vn \l_@@_hpos_block_str
7048             {
7049                 c { north }
7050                 l { north-west }
7051                 r { north-east }
7052             }
7053
7054         }
7055         B {
7056             \str_case:Vn \l_@@_hpos_block_str
7057             {
7058                 c { south }
7059                 l { south-west }
7060                 r { south-east }
7061             }
7062
7063         }
7064     }
7065 }
7066 \pgftransformshift
7067 {
7068     \pgfpointanchor
7069     {
7070         \@@_env: - #1 - #2 - block
7071         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7072     }
7073     { \l_tmpa_tl }
7074 }
7075 \pgfset
7076 {
7077     inner~xsep = \c_zero_dim ,
7078     inner~ysep = \l_@@_block_ysep_dim
7079 }
7080 \pgfnode
7081 { rectangle }
7082 { \l_tmpa_tl }
7083 { \box_use_drop:N \l_@@_cell_box } { } { }
7084 }
7085 {
7086     \pgfextracty \l_tmpa_dim
7087     {
7088         \@@_qpoint:n
7089         {
7090             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }

```

```

7091         - base
7092     }
7093 }
7094 \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.

```

7095 \pgfpointanchor
7096 {
7097     \@@_env: - #1 - #2 - block
7098     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7099 }
7100 {
7101     \str_case:Vn \l_@@_hpos_block_str
7102     {
7103         c { center }
7104         l { west }
7105         r { east }
7106     }
7107 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7108 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7109 \pgfset { inner~sep = \c_zero_dim }
7110 \pgfnode
7111 { rectangle }
7112 {
7113     \str_case:Vn \l_@@_hpos_block_str
7114     {
7115         c { base }
7116         l { base~west }
7117         r { base~east }
7118     }
7119 }
7120 { \box_use_drop:N \l_@@_cell_box } { } { }
7121 }
7122 \endpgfpicture
7123 \group_end:
7124 }

```

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).

```

7125 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7126 {
7127     \group_begin:
7128     \tl_clear:N \l_@@_draw_tl
7129     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7130     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7131     \pgfpicture
7132     \pgfrememberpicturepositiononpagetrue
7133     \pgf@relevantforpicturesizefalse
7134     \tl_if_empty:NF \l_@@_draw_tl
7135     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7136     \str_if_eq:VnTF \l_@@_draw_tl { default }
7137     { \CT@arc@ }
7138     { \@@_color:V \l_@@_draw_tl }
7139 }
7140 \pgfsetcornersarced
7141 {
7142     \pgfpoint

```

```

7143         { \dim_use:N \l_@@_rounded_corners_dim }
7144         { \dim_use:N \l_@@_rounded_corners_dim }
7145     }
7146     \@@_cut_on_hyphen:w #2 \q_stop
7147     \bool_lazy_and:nnT
7148     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7149     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7150     {
7151         \@@_qpoint:n { row - \l_tmpa_tl }
7152         \dim_set:Nn \l_tmpb_dim { \pgf@y }
7153         \@@_qpoint:n { col - \l_tmpb_tl }
7154         \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
7155         \@@_cut_on_hyphen:w #3 \q_stop
7156         \int_compare:nNnT \l_tmpa_tl > \c@iRow
7157         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7158         \int_compare:nNnT \l_tmpb_tl > \c@jCol
7159         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7160         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7161         \dim_set:Nn \l_tmpa_dim { \pgf@y }
7162         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7163         \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
7164         \pgfpathrectanglecorners
7165         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7166         { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7167         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7168         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7169         { \pgfusepathqstroke }
7170         { \pgfusepath { stroke } }
7171     }
7172     \endpgfpicture
7173     \group_end:
7174 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7175 \keys_define:nn { NiceMatrix / BlockStroke }
7176 {
7177     color .tl_set:N = \l_@@_draw_tl ,
7178     draw .tl_set:N = \l_@@_draw_tl ,
7179     draw .default:n = default ,
7180     line-width .dim_set:N = \l_@@_line_width_dim ,
7181     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7182     rounded-corners .default:n = 4 pt
7183 }

```

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).

```

7184 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7185 {
7186     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7187     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7188     \@@_cut_on_hyphen:w #2 \q_stop
7189     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7190     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7191     \@@_cut_on_hyphen:w #3 \q_stop
7192     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7193     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7194     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7195     {
7196         \use:x
7197         {
7198             \@@_vline:n
7199             {

```

```

7200         position = ##1 ,
7201         start = \l_@@_tmpc_tl ,
7202         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7203         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7204     }
7205 }
7206 }
7207 }
7208 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7209 {
7210     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7211     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7212     \@@_cut_on_hyphen:w #2 \q_stop
7213     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7214     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7215     \@@_cut_on_hyphen:w #3 \q_stop
7216     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7217     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7218     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7219     {
7220         \use:x
7221         {
7222             \@@_hline:n
7223             {
7224                 position = ##1 ,
7225                 start = \l_@@_tmpd_tl ,
7226                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
7227                 total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7228             }
7229         }
7230     }
7231 }

```

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).

```

7232 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7233 {
7234     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7235     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7236     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7237     { \@@_error:n { borders~forbidden } }
7238     {
7239         \tl_clear_new:N \l_@@_borders_tikz_tl
7240         \keys_set:nV
7241         { NiceMatrix / OnlyForTikzInBorders }
7242         \l_@@_borders_clist
7243         \@@_cut_on_hyphen:w #2 \q_stop
7244         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7245         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7246         \@@_cut_on_hyphen:w #3 \q_stop
7247         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7248         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7249         \@@_stroke_borders_block_i:
7250     }
7251 }
7252 \hook_gput_code:nnn { begindocument } { . }
7253 {
7254     \cs_new_protected:Npx \@@_stroke_borders_block_i:
7255     {
7256         \c_@@_pgfortikzpicture_tl
7257         \@@_stroke_borders_block_ii:

```

```

7258     \c_@@_endpgfortikzpicture_tl
7259   }
7260 }

7261 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7262 {
7263   \pgfrememberpicturepositiononpagetrue
7264   \pgf@relevantforpicturesizefalse
7265   \CT@arc@
7266   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7267   \clist_if_in:NnT \l_@@_borders_clist { right }
7268     { \@@_stroke_vertical:n \l_tmpb_tl }
7269   \clist_if_in:NnT \l_@@_borders_clist { left }
7270     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7271   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7272     { \@@_stroke_horizontal:n \l_tmpa_tl }
7273   \clist_if_in:NnT \l_@@_borders_clist { top }
7274     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7275 }

7276 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7277 {
7278   tikz .code:n =
7279     \cs_if_exist:NTF \tikzpicture
7280       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7281       { \@@_error:n { tikz-in~borders~without~tikz } } ,
7282   tikz .value_required:n = true ,
7283   top .code:n = ,
7284   bottom .code:n = ,
7285   left .code:n = ,
7286   right .code:n = ,
7287   unknown .code:n = \@@_error:n { bad~border }
7288 }

```

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).

```

7289 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7290 {
7291   \@@_qpoint:n \l_@@_tmpc_tl
7292   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7293   \@@_qpoint:n \l_tmpa_tl
7294   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7295   \@@_qpoint:n { #1 }
7296   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7297   {
7298     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7299     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7300     \pgfusepathqstroke
7301   }
7302   {
7303     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7304     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7305   }
7306 }

```

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).

```

7307 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7308 {
7309   \@@_qpoint:n \l_@@_tmpd_tl
7310   \clist_if_in:NnTF \l_@@_borders_clist { left }
7311     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7312     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7313   \@@_qpoint:n \l_tmpb_tl

```

```

7314 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7315 \@@_qpoint:n { #1 }
7316 \tl_if_empty:NTF \l_@@_borders_tikz_tl
7317 {
7318   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7319   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7320   \pgfusepathqstroke
7321 }
7322 {
7323   \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7324   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7325 }
7326 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7327 \keys_define:nn { NiceMatrix / BlockBorders }
7328 {
7329   borders .clist_set:N = \l_@@_borders_clist ,
7330   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7331   rounded-corners .default:n = 4 pt ,
7332   line-width .dim_set:N = \l_@@_line_width_dim ,
7333 }

```

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.

```

7334 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7335 {
7336   \begin { tikzpicture }
7337   \clist_map_inline:nn { #5 }
7338   {
7339     \path [ ##1 ]
7340       ( #1 -| #2 )
7341       rectangle
7342       ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7343   }
7344   \end { tikzpicture }
7345 }

```

## 27 How to draw the dotted lines transparently

```

7346 \cs_set_protected:Npn \@@_renew_matrix:
7347 {
7348   \RenewDocumentEnvironment { pmatrix } { } {
7349     { \pNiceMatrix }
7350     { \endpNiceMatrix }
7351   \RenewDocumentEnvironment { vmatrix } { } {
7352     { \vNiceMatrix }
7353     { \endvNiceMatrix }
7354   \RenewDocumentEnvironment { Vmatrix } { } {
7355     { \VNiceMatrix }
7356     { \endVNiceMatrix }
7357   \RenewDocumentEnvironment { bmatrix } { } {
7358     { \bNiceMatrix }
7359     { \endbNiceMatrix }
7360   \RenewDocumentEnvironment { Bmatrix } { } {
7361     { \BNiceMatrix }
7362     { \endBNiceMatrix }
7363 }

```



## 28 Automatic arrays

We will extract the potential keys `columns-type`, `l`, `c`, `r` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7364 \keys_define:nn { NiceMatrix / Auto }
7365 {
7366   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7367   columns-type .value_required:n = true ,
7368   l .meta:n = { columns-type = l } ,
7369   r .meta:n = { columns-type = r } ,
7370   c .meta:n = { columns-type = c } ,
7371   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7372   delimiters / color .value_required:n = true ,
7373   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7374   delimiters / max-width .default:n = true ,
7375   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7376   delimiters .value_required:n = true ,
7377 }

7378 \NewDocumentCommand \AutoNiceMatrixWithDelims
7379 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7380 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

7381 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7382 {

```

The group is for the protection of the keys.

```

7383 \group_begin:
7384 \bool_set_true:N \l_@@_Matrix_bool
7385 \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl

```

We nullify the command `\@@_transform_preamble:` because we will provide a preamble which is yet transformed (by using `\l_@@_columns_type_tl` which is yet `nicematrix-ready`).

```

7386 \cs_set_eq:NN \@@_transform_preamble: \prg_do_nothing:
7387 \use:x
7388 {
7389   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7390   { * { #4 } { \exp_not:N \l_@@_columns_type_tl } }
7391   [ \exp_not:N \l_tmpa_tl ]
7392 }
7393 \int_compare:nNnT \l_@@_first_row_int = 0
7394 {
7395   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7396   \prg_replicate:nn { #4 - 1 } { & }
7397   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7398 }
7399 \prg_replicate:nn { #3 }
7400 {
7401   \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

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`).

```

7402   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7403   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7404 }
7405 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7406 {
7407   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7408   \prg_replicate:nn { #4 - 1 } { & }
7409   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7410 }
7411 \end { NiceArrayWithDelims }
7412 \group_end:

```

```

7413 }
7414 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7415 {
7416   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7417   {
7418     \bool_gset_false:N \g_@@_NiceArray_bool
7419     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7420     \AutoNiceMatrixWithDelims { #2 } { #3 }
7421   }
7422 }
7423 \@@_define_com:nnn p ( )
7424 \@@_define_com:nnn b [ ]
7425 \@@_define_com:nnn v | |
7426 \@@_define_com:nnn V \l \l
7427 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7428 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7429 {
7430   \group_begin:
7431   \bool_gset_true:N \g_@@_NiceArray_bool
7432   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7433   \group_end:
7434 }

```

## 29 The redefinition of the command `\dotfill`

```

7435 \cs_set_eq:NN \@@_old_dotfill \dotfill
7436 \cs_new_protected:Npn \@@_dotfill:
7437 {

```

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}`).

```

7438   \@@_old_dotfill
7439   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7440 }

```

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`.

```

7441 \cs_new_protected:Npn \@@_dotfill_i:
7442 { \dim_compare:nNt { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

## 30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

7443 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7444 {
7445   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7446   {
7447     \@@_actually_diagbox:nnnnnn
7448     { \int_use:N \c_iRow }
7449     { \int_use:N \c_jCol }
7450     { \int_use:N \c_iRow }
7451     { \int_use:N \c_jCol }

```

```

7452         { \exp_not:n { #1 } }
7453         { \exp_not:n { #2 } }
7454     }

```

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`.

```

7455     \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7456     {
7457         { \int_use:N \c@iRow }
7458         { \int_use:N \c@jCol }
7459         { \int_use:N \c@iRow }
7460         { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7461         { }
7462     }
7463 }

```

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.

```

7464 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7465 {
7466     \pgfpicture
7467     \pgf@relevantforpicturesizefalse
7468     \pgfrememberpicturepositiononpagetrue
7469     \@@_qpoint:n { row - #1 }
7470     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7471     \@@_qpoint:n { col - #2 }
7472     \dim_set_eq:NN \l_tmpb_dim \pgf@x
7473     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7474     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7475     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7476     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7477     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7478     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7479     {

```

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.

```

7480         \CT@arc@
7481         \pgfsetroundcap
7482         \pgfusepathqstroke
7483     }
7484     \pgfset { inner~sep = 1 pt }
7485     \pgfscope
7486     \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7487     \pgfnode { rectangle } { south~west }
7488     {
7489         \begin { minipage } { 20 cm }
7490         \@@_math_toggle_token: #5 \@@_math_toggle_token:
7491         \end { minipage }
7492     }
7493     { }
7494     { }
7495 \endpgfscope
7496 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7497 \pgfnode { rectangle } { north~east }
7498 {
7499     \begin { minipage } { 20 cm }
7500     \raggedleft
7501     \@@_math_toggle_token: #6 \@@_math_toggle_token:
7502     \end { minipage }

```

```

7503     }
7504     { }
7505     { }
7506 \endpgfpicture
7507 }

```

## 31 The keyword `\CodeAfter`

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.

```

7508 \keys_define:nn { NiceMatrix }
7509 {
7510   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7511   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7512 }
7513 \keys_define:nn { NiceMatrix / CodeAfter }
7514 {
7515   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7516   sub-matrix .value_required:n = true ,
7517   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7518   delimiters / color .value_required:n = true ,
7519   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7520   rules .value_required:n = true ,
7521   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
7522 }

```

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. 77.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

7523 \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 `\`.

```

7524 \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`.

```

7525 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7526 {
7527   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7528   \@@_CodeAfter_iv:n
7529 }

```

We catch the argument of the command `\end` (in `#1`).

```

7530 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7531 {

```

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.

```

7532   \str_if_eq:eeTF \@currenvir { #1 }
7533   { \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`.

```

7534     {
7535         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7536         \@@_CodeAfter_ii:n
7537     }
7538 }
```

## 32 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

7539 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7540 {
7541     \pgfpicture
7542     \pgfrememberpicturepositiononpagetrue
7543     \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.

```

7544     \@@_qpoint:n { row - 1 }
7545     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7546     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7547     \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).

```

7548     \bool_if:nTF { #3 }
7549     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7550     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7551     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7552     {
7553         \cs_if_exist:cT
7554         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7555         {
7556             \pgfpointanchor
7557             { \@@_env: - ##1 - #2 }
7558             { \bool_if:nTF { #3 } { west } { east } }
7559             \dim_set:Nn \l_tmpa_dim
7560             { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7561         }
7562     }
```

Now we can put the delimiter with a node of PGF.

```

7563     \pgfset { inner~sep = \c_zero_dim }
7564     \dim_zero:N \nulldelimiterspace
7565     \pgftransformshift
7566     {
7567         \pgfpoint
7568         { \l_tmpa_dim }
```

```

7569         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7570     }
7571     \pgfnode
7572     { rectangle }
7573     { \bool_if:nTF { #3 } { east } { west } }
7574     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7575         \nullfont
7576         \c_math_toggle_token
7577         \@@_color:V \l_@@_delimiters_color_tl
7578         \bool_if:nTF { #3 } { \left #1 } { \left . }
7579         \vcenter
7580         {
7581             \nullfont
7582             \hrule \@height
7583                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7584                 \@depth \c_zero_dim
7585                 \@width \c_zero_dim
7586         }
7587         \bool_if:nTF { #3 } { \right . } { \right #1 }
7588         \c_math_toggle_token
7589     }
7590     { }
7591     { }
7592 \endpgfpicture
7593 }

```

### 33 The command `\SubMatrix`

```

7594 \keys_define:nn { NiceMatrix / sub-matrix }
7595 {
7596     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7597     extra-height .value_required:n = true ,
7598     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7599     left-xshift .value_required:n = true ,
7600     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7601     right-xshift .value_required:n = true ,
7602     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7603     xshift .value_required:n = true ,
7604     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7605     delimiters / color .value_required:n = true ,
7606     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7607     slim .default:n = true ,
7608     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7609     hlines .default:n = all ,
7610     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7611     vlines .default:n = all ,
7612     hvlines .meta:n = { hlines, vlines } ,
7613     hvlines .value_forbidden:n = true ,
7614 }
7615 \keys_define:nn { NiceMatrix }
7616 {
7617     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7618     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7619     NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7620     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7621     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7622     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7623 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

7624 \keys_define:nn { NiceMatrix / SubMatrix }
7625 {
7626   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7627   delimiters / color .value_required:n = true ,
7628   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7629   hlines .default:n = all ,
7630   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7631   vlines .default:n = all ,
7632   hvlines .meta:n = { hlines, vlines } ,
7633   hvlines .value_forbidden:n = true ,
7634   name .code:n =
7635     \tl_if_empty:nTF { #1 }
7636     { \@@_error:n { Invalid-name } }
7637     {
7638       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7639       {
7640         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7641         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7642         {
7643           \str_set:Nn \l_@@_submatrix_name_str { #1 }
7644           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7645         }
7646       }
7647       { \@@_error:n { Invalid-name } }
7648     } ,
7649   name .value_required:n = true ,
7650   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7651   rules .value_required:n = true ,
7652   code .tl_set:N = \l_@@_code_tl ,
7653   code .value_required:n = true ,
7654   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
7655 }

7656 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7657 {
7658   \peek_remove_spaces:n
7659   {
7660     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7661     {
7662       \SubMatrix { #1 } { #2 } { #3 } { #4 }
7663       [
7664         delimiters / color = \l_@@_delimiters_color_tl ,
7665         hlines = \l_@@_submatrix_hlines_clist ,
7666         vlines = \l_@@_submatrix_vlines_clist ,
7667         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
7668         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
7669         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
7670         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
7671         #5
7672       ]
7673     }
7674     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7675   }
7676 }

7677 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
7678 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7679 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

7680 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
7681 {
7682   \seq_gput_right:Nx \g_@@_submatrix_seq

```

```

7683     {
We use \str_if_eq:nnTF because it is fully expandable.
7684     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
7685     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
7686     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
7687     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
7688   }
7689 }

```

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.

```

7690 \hook_gput_code:nnn { begindocument } { . }
7691 {
7692   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
7693   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
7694   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
7695     {
7696       \peek_remove_spaces:n
7697       {
7698         \@@_sub_matrix:nnnnnnn
7699         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
7700       }
7701     }
7702 }

```

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).

```

7703 \NewDocumentCommand \@@_compute_i_j:nn
7704 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7705 { \@@_compute_i_j:nnnn #1 #2 }
7706 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
7707 {
7708   \tl_set:Nn \l_@@_first_i_tl { #1 }
7709   \tl_set:Nn \l_@@_first_j_tl { #2 }
7710   \tl_set:Nn \l_@@_last_i_tl { #3 }
7711   \tl_set:Nn \l_@@_last_j_tl { #4 }
7712   \tl_if_eq:NnT \l_@@_first_i_tl { last }
7713     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
7714   \tl_if_eq:NnT \l_@@_first_j_tl { last }
7715     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
7716   \tl_if_eq:NnT \l_@@_last_i_tl { last }
7717     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
7718   \tl_if_eq:NnT \l_@@_last_j_tl { last }
7719     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
7720 }

```



```

7721 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
7722 {
7723   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

7724   \@@_compute_i_j:nn { #2 } { #3 }
7725   \bool_lazy_or:nnTF
7726     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7727     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7728     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
7729     {
7730       \str_clear_new:N \l_@@_submatrix_name_str
7731       \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
7732       \pgfpicture
7733       \pgfrememberpicturepositiononpagetrue
7734       \pgf@relevantforpicturesizefalse
7735       \pgfset { inner-sep = \c_zero_dim }
7736       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7737       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int\_step\_inline:nnn is provided by currification.

```

7738   \bool_if:NTF \l_@@_submatrix_slim_bool
7739     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
7740     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
7741     {
7742       \cs_if_exist:cT
7743         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7744         {
7745           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7746           \dim_set:Nn \l_@@_x_initial_dim
7747             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7748         }
7749       \cs_if_exist:cT
7750         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7751         {
7752           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7753           \dim_set:Nn \l_@@_x_final_dim
7754             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7755         }
7756     }
7757   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7758     { \@@_error:nn { Impossible-delimiter } { left } }
7759     {
7760       \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7761         { \@@_error:nn { Impossible-delimiter } { right } }
7762         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7763     }
7764   \endpgfpicture
7765 }
7766 \group_end:
7767 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7768 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7769 {
7770   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7771   \dim_set:Nn \l_@@_y_initial_dim
7772     {
7773       \fp_to_dim:n
7774         {
7775           \pgf@y
7776           + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
7777         }
7778     } % modified 6.13c

```

```

7779 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7780 \dim_set:Nn \l_@@_y_final_dim
7781 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
7782 % modified 6.13c
7783 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7784 {
7785   \cs_if_exist:cT
7786   { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7787   {
7788     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7789     \dim_set:Nn \l_@@_y_initial_dim
7790     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7791   }
7792   \cs_if_exist:cT
7793   { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7794   {
7795     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7796     \dim_set:Nn \l_@@_y_final_dim
7797     { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7798   }
7799 }
7800 \dim_set:Nn \l_tmpa_dim
7801 {
7802   \l_@@_y_initial_dim - \l_@@_y_final_dim +
7803   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7804 }
7805 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7806 \group_begin:
7807 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7808 \@@_set_CT@arc@:V \l_@@_rules_color_tl
7809 \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`.

```

7810 \seq_map_inline:Nn \g_@@_cols_vlism_seq
7811 {
7812   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7813   {
7814     \int_compare:nNnT
7815     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
7816     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

7817 \@@_qpoint:n { col - ##1 }
7818 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7819 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7820 \pgfusepathqstroke
7821 }
7822 }
7823 }

```

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.

```

7824 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7825 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7826 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7827 {
7828   \bool_lazy_and:nnTF
7829   { \int_compare_p:nNn { ##1 } > 0 }
7830   {

```

```

7831         \int_compare_p:nNn
7832         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7833     {
7834         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7835         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7836         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7837         \pgfusepathqstroke
7838     }
7839     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
7840 }

```

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.

```

7841 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7842 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7843 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7844 {
7845     \bool_lazy_and:nnTF
7846     { \int_compare_p:nNn { ##1 } > 0 }
7847     {
7848         \int_compare_p:nNn
7849         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7850     {
7851         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7852     \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
7853     \dim_set:Nn \l_tmpa_dim
7854     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7855     \str_case:nn { #1 }
7856     {
7857         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7858         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7859         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7860     }
7861     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

7862     \dim_set:Nn \l_tmpb_dim
7863     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7864     \str_case:nn { #2 }
7865     {
7866         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7867         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7868         \{ { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7869     }
7870     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7871     \pgfusepathqstroke
7872     \group_end:
7873 }
7874 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7875 }

```

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).

```

7876 \str_if_empty:NF \l_@@_submatrix_name_str
7877 {
7878     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7879     \l_@@_x_initial_dim \l_@@_y_initial_dim
7880     \l_@@_x_final_dim \l_@@_y_final_dim
7881 }
7882 \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`.

```

7883 \begin { pgfscope }
7884 \pgftransformshift
7885 {
7886   \pgfpoint
7887   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7888   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7889 }
7890 \str_if_empty:NTF \l_@@_submatrix_name_str
7891 { \@@_node_left:nn #1 { } }
7892 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7893 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7894 \pgftransformshift
7895 {
7896   \pgfpoint
7897   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7898   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7899 }
7900 \str_if_empty:NTF \l_@@_submatrix_name_str
7901 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7902 {
7903   \@@_node_right:nnnn #2
7904   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7905 }
7906 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7907 \flag_clear_new:n { nicematrix }
7908 \l_@@_code_tl
7909 }

```

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  $i-lj$  refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

7910 \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.

```

7911 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
7912 {
7913   \use:e
7914   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
7915 }

```

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 -.

```

7916 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
7917 { #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.

```

7918 \tl_const:Nn \c_@@_integers_alist_tl
7919 {
7920   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7921   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7922   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7923   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7924 }

7925 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7926 {

```

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`.

```

7927   \tl_if_empty:nTF { #2 }
7928   {
7929     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7930     {
7931       \flag_raise:n { nicematrix }
7932       \int_if_even:nTF { \flag_height:n { nicematrix } }
7933       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7934       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7935     }
7936     { #1 }
7937   }

```

If there is an hyphen, we have to see whether we have a node of the form  $i-j$ , row- $i$  or col- $j$ .

```

7938   { \@@_pgfpointanchor_iii:w { #1 } #2 }
7939 }

```

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`).

```

7940 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
7941 {
7942   \str_case:nnF { #1 }
7943   {
7944     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7945     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7946   }

```

Now the case of a node of the form  $i-j$ .

```

7947   {
7948     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7949     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7950   }
7951 }

```

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`).

```

7952 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7953 {
7954   \pgfnode
7955   { rectangle }
7956   { east }
7957   {
7958     \nullfont

```

```

7959 \c_math_toggle_token
7960 \@@_color:V \l_@@_delimiters_color_tl
7961 \left #1
7962 \vcenter
7963 {
7964 \nullfont
7965 \hrule \@height \l_tmpa_dim
7966 \@depth \c_zero_dim
7967 \@width \c_zero_dim
7968 }
7969 \right .
7970 \c_math_toggle_token
7971 }
7972 { #2 }
7973 { }
7974 }

```

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.

```

7975 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
7976 {
7977 \pgfnode
7978 { rectangle }
7979 { west }
7980 {
7981 \nullfont
7982 \c_math_toggle_token
7983 \@@_color:V \l_@@_delimiters_color_tl
7984 \left .
7985 \vcenter
7986 {
7987 \nullfont
7988 \hrule \@height \l_tmpa_dim
7989 \@depth \c_zero_dim
7990 \@width \c_zero_dim
7991 }
7992 \right #1
7993 \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
7994 ^ { \smash { #4 } }
7995 \c_math_toggle_token
7996 }
7997 { #2 }
7998 { }
7999 }

```

## 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8000 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8001 {
8002 \peek_remove_spaces:n
8003 { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8004 }
8005 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8006 {
8007 \peek_remove_spaces:n
8008 { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8009 }

```

```

8010 \keys_define:nn { NiceMatrix / Brace }
8011 {
8012   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8013   left-shorten .default:n = true ,
8014   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8015   shorten .meta:n = { left-shorten , right-shorten } ,
8016   right-shorten .default:n = true ,
8017   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8018   yshift .value_required:n = true ,
8019   yshift .initial:n = \c_zero_dim ,
8020   color .tl_set:N = \l_tmpa_tl ,
8021   color .value_required:n = true ,
8022   unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
8023 }

```

#1 is the first cell of the rectangle (with the syntax  $i-lj$ ; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to under or over.

```

8024 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8025 {
8026   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8027   \@@_compute_i_j:nn { #1 } { #2 }
8028   \bool_lazy_or:nnTF
8029     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8030     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8031     {
8032       \str_if_eq:nnTF { #5 } { under }
8033       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8034       { \@@_error:nn { Construct~too~large } { \OverBrace } }
8035     }
8036     {
8037       \tl_clear:N \l_tmpa_tl
8038       \keys_set:nn { NiceMatrix / Brace } { #4 }
8039       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8040       \pgfpicture
8041       \pgfrememberpicturepositiononpagetrue
8042       \pgf@relevantforpicturesizefalse
8043       \bool_if:NT \l_@@_brace_left_shorten_bool
8044       {
8045         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8046         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8047         {
8048           \cs_if_exist:cT
8049             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8050             {
8051               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8052               \dim_set:Nn \l_@@_x_initial_dim
8053                 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8054             }
8055         }
8056       }
8057       \bool_lazy_or:nnT
8058         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8059         { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8060         {
8061           \@@_qpoint:n { col - \l_@@_first_j_tl }
8062           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8063         }
8064       \bool_if:NT \l_@@_brace_right_shorten_bool
8065       {
8066         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8067         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl

```

```

8068     {
8069         \cs_if_exist:cT
8070         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8071         {
8072             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8073             \dim_set:Nn \l_@@_x_final_dim
8074             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8075         }
8076     }
8077 }
8078 \bool_lazy_or:nnT
8079 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8080 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8081 {
8082     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8083     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8084 }
8085 \pgfset { inner~sep = \c_zero_dim }
8086 \str_if_eq:nnTF { #5 } { under }
8087 { \@@_underbrace_i:n { #3 } }
8088 { \@@_overbrace_i:n { #3 } }
8089 \endpgfpicture
8090 }
8091 \group_end:
8092 }

```

The argument is the text to put above the brace.

```

8093 \cs_new_protected:Npn \@@_overbrace_i:n #1
8094 {
8095     \@@_qpoint:n { row - \l_@@_first_i_tl }
8096     \pgftransformshift
8097     {
8098         \pgfpoint
8099         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8100         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8101     }
8102     \pgfnode
8103     { rectangle }
8104     { south }
8105     {
8106         \vbox_top:n
8107         {
8108             \group_begin:
8109             \everycr { }
8110             \halign
8111             {
8112                 \hfil ## \hfil \crrc
8113                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8114                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8115                 \c_math_toggle_token
8116                 \overbrace
8117                 {
8118                     \hbox_to_wd:nn
8119                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8120                     { }
8121                 }
8122                 \c_math_toggle_token
8123                 \cr
8124             }
8125             \group_end:
8126         }
8127     }
8128     { }
8129     { }

```



```
8130 }
```

The argument is the text to put under the brace.

```
8131 \cs_new_protected:Npn \@@_underbrace_i:n #1
8132 {
8133   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8134   \pgftransformshift
8135   {
8136     \pgfpoint
8137     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8138     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8139   }
8140   \pgfnode
8141   { rectangle }
8142   { north }
8143   {
8144     \group_begin:
8145     \everycr { }
8146     \vbox:n
8147     {
8148       \halign
8149       {
8150         \hfil ## \hfil \crrc
8151         \c_math_toggle_token
8152         \underbrace
8153         {
8154           \hbox_to_wd:nn
8155           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8156           { }
8157         }
8158         \c_math_toggle_token
8159         \cr
8160         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8161         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8162       }
8163     }
8164     \group_end:
8165   }
8166   { }
8167   { }
8168 }
```

## 35 The command \ShowCellNames

```
8169 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8170 {
8171   \dim_zero_new:N \g_@@_tmpc_dim
8172   \dim_zero_new:N \g_@@_tmpd_dim
8173   \dim_zero_new:N \g_@@_tmpe_dim
8174   \int_step_inline:nn \c@iRow
8175   {
8176     \begin { pgfpicture }
8177     \@@_qpoint:n { row - ##1 }
8178     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8179     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8180     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8181     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8182     \bool_if:NTF \l_@@_in_code_after_bool
8183     \end { pgfpicture }

```

```

8184 \int_step_inline:nn \c@jCol
8185 {
8186   \hbox_set:Nn \l_tmpa_box
8187     { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
8188   \begin { pgfpicture }
8189     \@@_qpoint:n { col - ####1 }
8190     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8191     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8192     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8193     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8194     \endpgfpicture
8195   \end { pgfpicture }
8196   \fp_set:Nn \l_tmpa_fp
8197     {
8198     \fp_min:nn
8199     {
8200       \fp_min:nn
8201       { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8202       { \dim_ratio:nn { \g_@@_tmpe_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8203     }
8204     { 1.0 }
8205   }
8206   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8207   \pgfpicture
8208   \pgfrememberpicturepositiononpagetrue
8209   \pgf@relevantforpicturesizefalse
8210   \pgftransformshift
8211   {
8212     \pgfpoint
8213     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8214     { \dim_use:N \g_@@_tmpa_dim }
8215   }
8216   \pgfnode
8217   { rectangle }
8218   { center }
8219   { \box_use:N \l_tmpa_box }
8220   { }
8221   { }
8222   \endpgfpicture
8223 }
8224 }
8225 }
8226 \NewDocumentCommand \@@_ShowCellNames { }
8227 {
8228   \bool_if:NT \l_@@_in_code_after_bool
8229   {
8230     \pgfpicture
8231     \pgfrememberpicturepositiononpagetrue
8232     \pgf@relevantforpicturesizefalse
8233     \pgfpathrectanglecorners
8234     { \@@_qpoint:n { 1 } }
8235     {
8236       \@@_qpoint:n
8237       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8238     }
8239     \pgfsetfillopacity { 0.75 }
8240     \pgfsetfillcolor { white }
8241     \pgfusepathqfill
8242     \endpgfpicture
8243   }
8244   \dim_zero_new:N \g_@@_tmpc_dim
8245   \dim_zero_new:N \g_@@_tmpd_dim
8246   \dim_zero_new:N \g_@@_tmpe_dim

```

```

8247 \int_step_inline:nn \c@iRow
8248 {
8249   \bool_if:NTF \l_@@_in_code_after_bool
8250   {
8251     \pgfpicture
8252     \pgfrememberpicturepositiononpagetrue
8253     \pgf@relevantforpicturesizefalse
8254   }
8255   { \begin { pgfpicture } }
8256   \@@_qpoint:n { row - ##1 }
8257   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8258   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8259   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8260   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8261   \bool_if:NTF \l_@@_in_code_after_bool
8262   { \endpgfpicture }
8263   { \end { pgfpicture } }
8264   \int_step_inline:nn \c@jCol
8265   {
8266     \hbox_set:Nn \l_tmpa_box
8267     {
8268       \normalfont \Large \sffamily \bfseries
8269       \bool_if:NTF \l_@@_in_code_after_bool
8270       { \color { red } }
8271       { \color { red ! 50 } }
8272       ##1 - #####1
8273     }
8274     \bool_if:NTF \l_@@_in_code_after_bool
8275     {
8276       \pgfpicture
8277       \pgfrememberpicturepositiononpagetrue
8278       \pgf@relevantforpicturesizefalse
8279     }
8280     { \begin { pgfpicture } }
8281     \@@_qpoint:n { col - #####1 }
8282     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8283     \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8284     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8285     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8286     \bool_if:NTF \l_@@_in_code_after_bool
8287     { \endpgfpicture }
8288     { \end { pgfpicture } }
8289     \fp_set:Nn \l_tmpa_fp
8290     {
8291       \fp_min:nn
8292       {
8293         \fp_min:nn
8294         { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8295         { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8296       }
8297       { 1.0 }
8298     }
8299     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8300     \pgfpicture
8301     \pgfrememberpicturepositiononpagetrue
8302     \pgf@relevantforpicturesizefalse
8303     \pgftransformshift
8304     {
8305       \pgfpoint
8306       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8307       { \dim_use:N \g_tmpa_dim }
8308     }
8309     \pgfnode

```

```

8310         { rectangle }
8311         { center }
8312         { \box_use:N \l_tmpa_box }
8313         { }
8314         { }
8315     \endpgfpicture
8316 }
8317 }
8318 }

```

## 36 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.

```

8319 \bool_new:N \c_@@_footnotehyper_bool

```

The boolean `\c_@@_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.

```

8320 \bool_new:N \c_@@_footnote_bool
8321 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
8322 {
8323     The~key~'\l_keys_key_str'~is~unknown.  \
8324     That~key~will~be~ignored.  \
8325     For~a~list~of~the~available~keys,~type~H~<return>.
8326 }
8327 {
8328     The~available~keys~are~(in~alphabetic~order):~
8329     footnote,~
8330     footnotehyper,~
8331     messages-for-Overleaf,~
8332     no-test-for-array,~
8333     renew-dots,~and
8334     renew-matrix.
8335 }
8336 \keys_define:nn { NiceMatrix / Package }
8337 {
8338     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8339     renew-dots .value_forbidden:n = true ,
8340     renew-matrix .code:n = \@@_renew_matrix: ,
8341     renew-matrix .value_forbidden:n = true ,
8342     messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
8343     footnote .bool_set:N = \c_@@_footnote_bool ,
8344     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
8345     no-test-for-array .bool_set:N = \c_@@_no_test_for_array_bool ,
8346     no-test-for-array .default:n = true ,
8347     unknown .code:n = \@@_error:n { Unknown~key~for~package }
8348 }
8349 \ProcessKeysOptions { NiceMatrix / Package }

8350 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8351 {
8352     You~can't~use~the~option~'footnote'~because~the~package~
8353     footnotehyper~has~already~been~loaded.~
8354     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~

```

```

8355   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8356   of~the~package~footnotehyper.\\
8357   The~package~footnote~won't~be~loaded.
8358 }

8359 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8360 {
8361   You~can't~use~the~option~'footnotehyper'~because~the~package~
8362   footnote~has~already~been~loaded.~
8363   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8364   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8365   of~the~package~footnote.\\
8366   The~package~footnotehyper~won't~be~loaded.
8367 }

```

```

8368 \bool_if:NT \c_@@_footnote_bool
8369 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8370   \IfClassLoadedTF { beamer }
8371   { \bool_set_false:N \c_@@_footnote_bool }
8372   {
8373     \IfPackageLoadedTF { footnotehyper }
8374     { \@@_error:n { footnote~with~footnotehyper~package } }
8375     { \usepackage { footnote } }
8376   }
8377 }

8378 \bool_if:NT \c_@@_footnotehyper_bool
8379 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8380   \IfClassLoadedTF { beamer }
8381   { \bool_set_false:N \c_@@_footnote_bool }
8382   {
8383     \IfPackageLoadedTF { footnote }
8384     { \@@_error:n { footnotehyper~with~footnote~package } }
8385     { \usepackage { footnotehyper } }
8386   }
8387   \bool_set_true:N \c_@@_footnote_bool
8388 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

## 37 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.

```

8389 \bool_new:N \l_@@_underscore_loaded_bool
8390 \IfPackageLoadedTF { underscore }
8391 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8392 { }

8393 \hook_gput_code:nnn { begindocument } { . }
8394 {
8395   \bool_if:NF \l_@@_underscore_loaded_bool
8396   {
8397     \IfPackageLoadedTF { underscore }

```

```

8398         { \@@_error:n { underscore~after~nicematrix } }
8399     { }
8400 }
8401 }

```

## 38 Error messages of the package

```

8402 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
8403 { \str_const:Nn \c_@@_available_keys_str { } }
8404 {
8405     \str_const:Nn \c_@@_available_keys_str
8406     { For~a~list~of~the~available~keys,~type~H~<return>. }
8407 }
8408 \seq_new:N \g_@@_types_of_matrix_seq
8409 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8410 {
8411     NiceMatrix ,
8412     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8413 }
8414 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8415 { \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:NVTF` 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`.

```

8416 \cs_new_protected:Npn \@@_error_too_much_cols:
8417 {
8418     \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8419     {
8420         \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8421         { \@@_fatal:n { too~much~cols~for~matrix } }
8422         {
8423             \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8424             { \@@_fatal:n { too~much~cols~for~matrix } }
8425             {
8426                 \bool_if:NF \l_@@_last_col_without_value_bool
8427                 { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
8428             }
8429         }
8430     }
8431     {
8432         \IfPackageLoadedTF { tabularx }
8433         {
8434             \str_if_eq:VnTF \g_@@_name_env_str { NiceTabularX }
8435             {
8436                 \int_compare:nNnTF \c@iRow = \c_zero_int
8437                 { \@@_fatal:n { X~columns~with~tabularx } }
8438                 {
8439                     \@@_fatal:nn { too~much~cols~for~array }
8440                     {
8441                         However,~this~message~may~be~erroneous:~
8442                         maybe~you~have~used~X~columns~while~'tabularx'~is~loaded,~
8443                         ~which~is~forbidden~(however,~it's~still~possible~to~use~
8444                         X~columns~in~{NiceTabularX}).
8445                     }
8446                 }
8447             }
8448         }
8449     }
8450 }

```

```

8448         { \@@_fatal:nn { too-much-cols-for-array } { } }
8449     }
8450     { \@@_fatal:nn { too-much-cols-for-array } { } }
8451 }
8452 }

```

The following command must *not* be protected since it's used in an error message.

```

8453 \cs_new:Npn \@@_message_hdotsfor:
8454 {
8455     \tl_if_empty:VF \g_@@_HVDotsfor_lines_tl
8456     { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
8457 }
8458 \@@_msg_new:nn { negative-weight }
8459 {
8460     Negative-weight.\
8461     The-weight-of-the-'X'~columns-must-be-positive-and-you-have-used~
8462     the-value-\int_use:N \l_@@_weight_int'.\
8463     The-absolute-value-will-be-used.
8464 }
8465 \@@_msg_new:nn { last-col-not-used }
8466 {
8467     Column-not-used.\
8468     The-key-'last-col'-is-in-force-but-you-have-not-used-that-last-column~
8469     in-your-\@@_full_name_env:~However,~you-can-go-on.
8470 }
8471 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
8472 {
8473     Too-much-columns.\
8474     In-the-row-\int_eval:n { \c@iRow },~
8475     you-try-to-use-more-columns~
8476     than-allowed-by-your-\@@_full_name_env:.\@@_message_hdotsfor:\
8477     The-maximal-number-of-columns-is-\int_eval:n { \l_@@_last_col_int - 1 }~
8478     (plus-the-exterior-columns).~This-error-is-fatal.
8479 }
8480 \@@_msg_new:nn { too-much-cols-for-matrix }
8481 {
8482     Too-much-columns.\
8483     In-the-row-\int_eval:n { \c@iRow },~
8484     you-try-to-use-more-columns-than-allowed-by-your~
8485     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
8486     number-of-columns-for-a-matrix~(excepted-the-potential-exterior~
8487     columns)~is-fixed-by-the-LaTeX-counter-'MaxMatrixCols'.~
8488     Its-current-value-is-\int_use:N \c@MaxMatrixCols\ (use~
8489     \token_to_str:N \setcounter\ to-change-that-value).~
8490     This-error-is-fatal.
8491 }
8492 \@@_msg_new:nn { too-much-cols-for-array }
8493 {
8494     Too-much-columns.\
8495     In-the-row-\int_eval:n { \c@iRow },~
8496     ~you-try-to-use-more-columns-than-allowed-by-your~
8497     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
8498     \int_use:N \g_@@_static_num_of_col_int\
8499     ~plus-the-potential-exterior-ones).~#1
8500     This-error-is-fatal.
8501 }
8502 \@@_msg_new:nn { X-columns-with-tabularx }
8503 {
8504     There-is-a-problem.\
8505     You-have-probably-used-X-columns-in-your-environment~{\g_@@_name_env_str}.~
8506     That's-not-allowed-because-'tabularx'-is-loaded~(however,~you-can-use-X-columns~

```

```

8507     in~an~environment~{NiceTabularX}).\\
8508     This~error~is~fatal.
8509 }

8510 \@@_msg_new:nn { columns~not~used }
8511 {
8512     Columns~not~used.\\
8513     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8514     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
8515     The~columns~you~did~not~used~won't~be~created.\\
8516     We~won't~have~similar~error~till~the~end~of~the~document.
8517 }

8518 \@@_msg_new:nn { in~first~col }
8519 {
8520     Erroneous~use.\\
8521     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
8522     That~command~will~be~ignored.
8523 }

8524 \@@_msg_new:nn { in~last~col }
8525 {
8526     Erroneous~use.\\
8527     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
8528     That~command~will~be~ignored.
8529 }

8530 \@@_msg_new:nn { in~first~row }
8531 {
8532     Erroneous~use.\\
8533     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
8534     That~command~will~be~ignored.
8535 }

8536 \@@_msg_new:nn { in~last~row }
8537 {
8538     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
8539     That~command~will~be~ignored.
8540 }

8541 \@@_msg_new:nn { caption~outside~float }
8542 {
8543     Key~caption~forbidden.\\
8544     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
8545     environment.~This~key~will~be~ignored.
8546 }

8547 \@@_msg_new:nn { short~caption~without~caption }
8548 {
8549     You~should~not~use~the~key~'short~caption'~without~'caption'.~
8550     However,~your~'short~caption'~will~be~used~as~'caption'.
8551 }

8552 \@@_msg_new:nn { double~closing~delimiter }
8553 {
8554     Double~delimiter.\\
8555     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
8556     delimiter.~This~delimiter~will~be~ignored.
8557 }

8558 \@@_msg_new:nn { delimiter~after~opening }
8559 {
8560     Double~delimiter.\\
8561     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
8562     delimiter.~That~delimiter~will~be~ignored.
8563 }

8564 \@@_msg_new:nn { bad~option~for~line~style }
8565 {

```



```

8566 Bad~line~style.\\
8567 Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
8568 is~'standard'.~That~key~will~be~ignored.
8569 }

8570 \@@_msg_new:nn { Identical~notes~in~caption }
8571 {
8572   Identical~tabular~notes.\\
8573   You~can't~put~several~notes~with~the~same~content~in~
8574   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
8575   If~you~go~on,~the~output~will~probably~be~erroneous.
8576 }

8577 \@@_msg_new:nn { tabularnote~below~the~tabular }
8578 {
8579   \token_to_str:N \tabularnote\ forbidden\\
8580   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8581   of~your~tabular~because~the~caption~will~be~composed~below~
8582   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8583   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
8584   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8585   no~similar~error~will~raised~in~this~document.
8586 }

8587 \@@_msg_new:nn { Unknown~key~for~rules }
8588 {
8589   Unknown~key.\\
8590   There~is~only~two~keys~available~here:~width~and~color.\\
8591   You~key~'\l_keys_key_str'~will~be~ignored.
8592 }

8593 \@@_msg_new:nnn { Unknown~key~for~custom~line }
8594 {
8595   Unknown~key.\\
8596   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
8597   It~you~go~on,~you~will~probably~have~other~errors. \\
8598   \c_@@_available_keys_str
8599 }
8600 {
8601   The~available~keys~are~(in~alphabetic~order):~
8602   ccommand,~
8603   color,~
8604   command,~
8605   dotted,~
8606   letter,~
8607   multiplicity,~
8608   sep-color,~
8609   tikz,~and~total-width.
8610 }

8611 \@@_msg_new:nnn { Unknown~key~for~xdots }
8612 {
8613   Unknown~key.\\
8614   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
8615   \c_@@_available_keys_str
8616 }
8617 {
8618   The~available~keys~are~(in~alphabetic~order):~
8619   'color',~
8620   'inter',~
8621   'line-style',~
8622   'radius',~
8623   'shorten',~
8624   'shorten-end'~and~'shorten-start'.
8625 }

8626 \@@_msg_new:nn { Unknown~key~for~rowcolors }

```

```

8627 {
8628   Unknown~key.\\
8629   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
8630   (and~you~try~to~use~'\l_keys_key_str')\\
8631   That~key~will~be~ignored.
8632 }
8633 \@@_msg_new:nn { label~without~caption }
8634 {
8635   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
8636   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
8637 }
8638 \@@_msg_new:nn { W~warning }
8639 {
8640   Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
8641   (row~\int_use:N \c@iRow).
8642 }
8643 \@@_msg_new:nn { Construct~too~large }
8644 {
8645   Construct~too~large.\\
8646   Your~command~\token_to_str:N #1
8647   can't~be~drawn~because~your~matrix~is~too~small.\\
8648   That~command~will~be~ignored.
8649 }
8650 \@@_msg_new:nn { underscore~after~nicematrix }
8651 {
8652   Problem~with~'underscore'.\\
8653   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
8654   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
8655   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
8656 }
8657 \@@_msg_new:nn { ampersand~in~light-syntax }
8658 {
8659   Ampersand~forbidden.\\
8660   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
8661   the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
8662 }
8663 \@@_msg_new:nn { double~backslash~in~light-syntax }
8664 {
8665   Double~backslash~forbidden.\\
8666   You~can't~use~\token_to_str:N
8667   \\~to~separate~rows~because~the~key~'light-syntax'~
8668   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
8669   (set~by~the~key~'end-of-row').~This~error~is~fatal.
8670 }
8671 \@@_msg_new:nn { hlines~with~color }
8672 {
8673   Incompatible~keys.\\
8674   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
8675   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
8676   Maybe~it~will~possible~in~future~version.\\
8677   Your~key~will~be~discarded.
8678 }
8679 \@@_msg_new:nn { bad~value~for~baseline }
8680 {
8681   Bad~value~for~baseline.\\
8682   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
8683   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
8684   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
8685   the~form~'line-i'.\\
8686   A~value~of~1~will~be~used.

```

```

8687     }
8688 \@@_msg_new:nn { ragged2e~not~loaded }
8689 {
8690     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
8691     your~column~'\l_@@_vpos_col_str'~(or~'X')~.~The~key~'\str_lowercase:V
8692     \l_keys_key_str'~will~be~used~instead.
8693 }
8694 \@@_msg_new:nn { Invalid~name }
8695 {
8696     Invalid~name.\\
8697     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
8698     \SubMatrix\ of~your~\@@_full_name_env:.\\
8699     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
8700     This~key~will~be~ignored.
8701 }
8702 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
8703 {
8704     Wrong~line.\\
8705     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
8706     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
8707     number~is~not~valid.~It~will~be~ignored.
8708 }
8709 \@@_msg_new:nn { Impossible~delimiter }
8710 {
8711     Impossible~delimiter.\\
8712     It's~impossible~to~draw~the~#1~delimiter~of~your~
8713     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
8714     in~that~column.
8715     \bool_if:NT \l_@@_submatrix_slim_bool
8716     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
8717     This~\token_to_str:N \SubMatrix\ will~be~ignored.
8718 }
8719 \@@_msg_new:nn { width~without~X~columns }
8720 {
8721     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
8722     That~key~will~be~ignored.
8723 }
8724 \@@_msg_new:nn { key~multiplicity~with~dotted }
8725 {
8726     Incompatible~keys. \\
8727     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
8728     in~a~'custom-line'.~They~are~incompatible. \\
8729     The~key~'multiplicity'~will~be~discarded.
8730 }
8731 \@@_msg_new:nn { empty~environment }
8732 {
8733     Empty~environment.\\
8734     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
8735 }
8736 \@@_msg_new:nn { No~letter~and~no~command }
8737 {
8738     Erroneous~use.\\
8739     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
8740     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
8741     '~ccommand'~(to~draw~horizontal~rules).\\
8742     However,~you~can~go~on.
8743 }
8744 \@@_msg_new:nn { Forbidden~letter }
8745 {
8746     Forbidden~letter.\\

```

```

8747     You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
8748     It~will~be~ignored.
8749 }

8750 \@@_msg_new:nn { Several~letters }
8751 {
8752     Wrong~name.\\
8753     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
8754     have~used~'\l_@@_letter_str').\\
8755     It~will~be~ignored.
8756 }

8757 \@@_msg_new:nn { Delimiter~with~small }
8758 {
8759     Delimiter~forbidden.\\
8760     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
8761     because~the~key~'small'~is~in~force.\\
8762     This~error~is~fatal.
8763 }

8764 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
8765 {
8766     Unknown~cell.\\
8767     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
8768     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
8769     can't~be~executed~because~a~cell~doesn't~exist.\\
8770     This~command~\token_to_str:N \line\ will~be~ignored.
8771 }

8772 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
8773 {
8774     Duplicate~name.\\
8775     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
8776     in~this~\@@_full_name_env:..\\
8777     This~key~will~be~ignored.\\
8778     \bool_if:NF \c_@@_messages_for_Overleaf_bool
8779     { For~a~list~of~the~names~already~used,~type~H~<return>. }
8780 }
8781 {
8782     The~names~already~defined~in~this~\@@_full_name_env:~ are:~
8783     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
8784 }

8785 \@@_msg_new:nn { r~or~l~with~preamble }
8786 {
8787     Erroneous~use.\\
8788     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:..
8789     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
8790     your~\@@_full_name_env:..\\
8791     This~key~will~be~ignored.
8792 }

8793 \@@_msg_new:nn { Hdotsfor~in~col~0 }
8794 {
8795     Erroneous~use.\\
8796     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
8797     the~array.~This~error~is~fatal.
8798 }

8799 \@@_msg_new:nn { bad~corner }
8800 {
8801     Bad~corner.\\
8802     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
8803     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
8804     This~specification~of~corner~will~be~ignored.
8805 }

8806 \@@_msg_new:nn { bad~border }

```

```

8807 {
8808   Bad~border.\\
8809   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
8810   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
8811   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
8812   also~use~the~key~'tikz'
8813   \IfPackageLoadedTF { tikz }
8814     { }
8815     {~if~you~load~the~LaTeX~package~'tikz'})\\.\\
8816   This~specification~of~border~will~be~ignored.
8817 }
8818 \@@_msg_new:nn { tikz~key~without~tikz }
8819 {
8820   Tikz~not~loaded.\\
8821   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
8822   \Block'~because~you~have~not~loaded~tikz.~
8823   This~key~will~be~ignored.
8824 }
8825 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
8826 {
8827   Erroneous~use.\\
8828   In~the~\@@_full_name_env:,~you~must~use~the~key~
8829   'last-col'~without~value.\\
8830   However,~you~can~go~on~for~this~time~
8831   (the~value~'\l_keys_value_tl'~will~be~ignored).
8832 }
8833 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
8834 {
8835   Erroneous~use.\\
8836   In~\NiceMatrixoptions,~you~must~use~the~key~
8837   'last-col'~without~value.\\
8838   However,~you~can~go~on~for~this~time~
8839   (the~value~'\l_keys_value_tl'~will~be~ignored).
8840 }
8841 \@@_msg_new:nn { Block~too~large~1 }
8842 {
8843   Block~too~large.\\
8844   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
8845   too~small~for~that~block. \\
8846 }
8847 \@@_msg_new:nn { Block~too~large~2 }
8848 {
8849   Block~too~large.\\
8850   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8851   \g_@@_static_num_of_col_int\
8852   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
8853   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
8854   (&)~at~the~end~of~the~first~row~of~your~
8855   \@@_full_name_env:.\\
8856   This~block~and~maybe~others~will~be~ignored.
8857 }
8858 \@@_msg_new:nn { unknown~column~type }
8859 {
8860   Bad~column~type.\\
8861   The~column~type~'#1'~in~your~\@@_full_name_env:\
8862   is~unknown. \\
8863   This~error~is~fatal.
8864 }
8865 \@@_msg_new:nn { unknown~column~type~S }
8866 {
8867   Bad~column~type.\\

```

```

8868 The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
8869 If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
8870 load~that~package. \\
8871 This~error~is~fatal.
8872 }

8873 \@@_msg_new:nn { tabularnote~forbidden }
8874 {
8875   Forbidden~command.\\
8876   You~can't~use~the~command~\token_to_str:N\tabularnote\
8877   ~here.~This~command~is~available~only~in~
8878   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
8879   the~argument~of~a~command~\token_to_str:N \caption\ included~
8880   in~an~environment~{table}. \\
8881   This~command~will~be~ignored.
8882 }

8883 \@@_msg_new:nn { borders~forbidden }
8884 {
8885   Forbidden~key.\\
8886   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
8887   because~the~option~'rounded~corners'~
8888   is~in~force~with~a~non~zero~value.\\
8889   This~key~will~be~ignored.
8890 }

8891 \@@_msg_new:nn { bottomrule~without~booktabs }
8892 {
8893   booktabs~not~loaded.\\
8894   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
8895   loaded~'booktabs'.\\
8896   This~key~will~be~ignored.
8897 }

8898 \@@_msg_new:nn { enumitem~not~loaded }
8899 {
8900   enumitem~not~loaded.\\
8901   You~can't~use~the~command~\token_to_str:N\tabularnote\
8902   ~because~you~haven't~loaded~'enumitem'.\\
8903   All~the~commands~\token_to_str:N\tabularnote\ will~be~
8904   ignored~in~the~document.
8905 }

8906 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
8907 {
8908   Tikz~not~loaded.\\
8909   You~have~used~the~key~'tikz'~in~the~definition~of~a~
8910   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
8911   You~can~go~on~but~you~will~have~another~error~if~you~actually~
8912   use~that~custom~line.
8913 }

8914 \@@_msg_new:nn { tikz~in~borders~without~tikz }
8915 {
8916   Tikz~not~loaded.\\
8917   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
8918   command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
8919   That~key~will~be~ignored.
8920 }

8921 \@@_msg_new:nn { color~in~custom~line~with~tikz }
8922 {
8923   Erroneous~use.\\
8924   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
8925   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
8926   The~key~'color'~will~be~discarded.
8927 }

```

```

8928 \@@_msg_new:nn { Wrong-last-row }
8929 {
8930   Wrong-number.\
8931   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
8932   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
8933   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
8934   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
8935   without~value~(more~compilations~might~be~necessary).
8936 }
8937 \@@_msg_new:nn { Yet~in~env }
8938 {
8939   Nested~environments.\
8940   Environments~of~nicematrix~can't~be~nested.\
8941   This~error~is~fatal.
8942 }
8943 \@@_msg_new:nn { Outside-math-mode }
8944 {
8945   Outside~math~mode.\
8946   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
8947   (and~not~in~\token_to_str:N \vcenter).\
8948   This~error~is~fatal.
8949 }
8950 \@@_msg_new:nn { One~letter~allowed }
8951 {
8952   Bad~name.\
8953   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
8954   It~will~be~ignored.
8955 }
8956 \@@_msg_new:nn { TabularNote~in~CodeAfter }
8957 {
8958   Environment~{TabularNote}~forbidden.\
8959   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
8960   but~*before*~the~\token_to_str:N \CodeAfter.\
8961   This~environment~{TabularNote}~will~be~ignored.
8962 }
8963 \@@_msg_new:nn { varwidth~not~loaded }
8964 {
8965   varwidth~not~loaded.\
8966   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
8967   loaded.\
8968   Your~column~will~behave~like~'p'.
8969 }
8970 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
8971 {
8972   Unknow~key.\
8973   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
8974   \c_@@_available_keys_str
8975 }
8976 {
8977   The~available~keys~are~(in~alphabetic~order):~
8978   color,~
8979   dotted,~
8980   multiplicity,~
8981   sep-color,~
8982   tikz,~and~total~width.
8983 }
8984
8985 \@@_msg_new:nnn { Unknown~key~for~Block }
8986 {
8987   Unknown~key.\
8988   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N

```

```

8989 \Block.\ It~will~be~ignored. \
8990 \c_@@_available_keys_str
8991 }
8992 {
8993 The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
8994 hlines,~hvlines,~l,~line~width,~name,~rounded~corners,~r,~respect~arraystretch,~
8995 t,~T,~tikz,~transparent~and~vlines.
8996 }
8997 \@@_msg_new:nn { Version~of~siunitx~too~old }
8998 {
8999 siunitx~too~old.\
9000 You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9001 is~too~old.~You~need~at~least~v~3.0.38~and~your~log~file~says:~"siunitx,~
9002 \use:c { ver @ siunitx.sty }". \
9003 This~error~is~fatal.
9004 }
9005 \@@_msg_new:nnn { Unknown~key~for~Brace }
9006 {
9007 Unknown~key.\
9008 The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9009 \UnderBrace\ and~\token_to_str:N \OverBrace.\
9010 It~will~be~ignored. \
9011 \c_@@_available_keys_str
9012 }
9013 {
9014 The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9015 right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9016 right~shorten)~and~yshift.
9017 }
9018 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9019 {
9020 Unknown~key.\
9021 The~key~'\l_keys_key_str'~is~unknown.\
9022 It~will~be~ignored. \
9023 \c_@@_available_keys_str
9024 }
9025 {
9026 The~available~keys~are~(in~alphabetic~order):~
9027 delimiters/color,~
9028 rules~(with~the~subkeys~'color'~and~'width'),~
9029 sub~matrix~(several~subkeys)~
9030 and~xdots~(several~subkeys).~
9031 The~latter~is~for~the~command~\token_to_str:N \line.
9032 }
9033 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9034 {
9035 Unknown~key.\
9036 The~key~'\l_keys_key_str'~is~unknown.\
9037 It~will~be~ignored. \
9038 \c_@@_available_keys_str
9039 }
9040 {
9041 The~available~keys~are~(in~alphabetic~order):~
9042 create~cell~nodes,~
9043 delimiters/color~and~
9044 sub~matrix~(several~subkeys).
9045 }
9046 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9047 {
9048 Unknown~key.\
9049 The~key~'\l_keys_key_str'~is~unknown.\
9050 That~key~will~be~ignored. \

```



```

9051 \c_@@_available_keys_str
9052 }
9053 {
9054 The~available~keys~are~(in~alphabetic~order):~
9055 'delimiters/color',~
9056 'extra-height',~
9057 'hlines',~
9058 'hvlines',~
9059 'left-xshift',~
9060 'name',~
9061 'right-xshift',~
9062 'rules'~(with~the~subkeys~'color'~and~'width'),~
9063 'slim',~
9064 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9065 and~'right-xshift').\\
9066 }
9067 \@@_msg_new:nnn { Unknown~key~for~notes }
9068 {
9069 Unknown~key.\\
9070 The~key~'\l_keys_key_str'~is~unknown.\\
9071 That~key~will~be~ignored. \\
9072 \c_@@_available_keys_str
9073 }
9074 {
9075 The~available~keys~are~(in~alphabetic~order):~
9076 bottomrule,~
9077 code-after,~
9078 code-before,~
9079 detect-duplicates,~
9080 enumitem-keys,~
9081 enumitem-keys-para,~
9082 para,~
9083 label-in-list,~
9084 label-in-tabular~and~
9085 style.
9086 }
9087 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9088 {
9089 Unknown~key.\\
9090 The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9091 \token_to_str:N \RowStyle. \\
9092 That~key~will~be~ignored. \\
9093 \c_@@_available_keys_str
9094 }
9095 {
9096 The~available~keys~are~(in~alphabetic~order):~
9097 'bold',~
9098 'cell-space-top-limit',~
9099 'cell-space-bottom-limit',~
9100 'cell-space-limits',~
9101 'color',~
9102 'nb-rows'~and~
9103 'rowcolor'.
9104 }
9105 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9106 {
9107 Unknown~key.\\
9108 The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9109 \token_to_str:N \NiceMatrixOptions. \\
9110 That~key~will~be~ignored. \\
9111 \c_@@_available_keys_str
9112 }
9113 {

```

```

9114 The~available~keys~are~(in~alphabetic~order):~
9115 allow~duplicate~names,~
9116 caption~above,~
9117 cell~space~bottom~limit,~
9118 cell~space~limits,~
9119 cell~space~top~limit,~
9120 code~for~first~col,~
9121 code~for~first~row,~
9122 code~for~last~col,~
9123 code~for~last~row,~
9124 corners,~
9125 custom~key,~
9126 create~extra~nodes,~
9127 create~medium~nodes,~
9128 create~large~nodes,~
9129 delimiters~(several~subkeys),~
9130 end~of~row,~
9131 first~col,~
9132 first~row,~
9133 hlines,~
9134 hvlines,~
9135 hvlines~except~borders,~
9136 last~col,~
9137 last~row,~
9138 left~margin,~
9139 light~syntax,~
9140 matrix/columns~type,~
9141 notes~(several~subkeys),~
9142 nullify~dots,~
9143 pgf~node~code,~
9144 renew~dots,~
9145 renew~matrix,~
9146 respect~arraystretch,~
9147 right~margin,~
9148 rules~(with~the~subkeys~'color'~and~'width'),~
9149 small,~
9150 sub~matrix~(several~subkeys),~
9151 vlines,~
9152 xdots~(several~subkeys).
9153 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9154 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9155 {
9156   Unknown~key.\\
9157   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9158   \{NiceArray\}. \\
9159   That~key~will~be~ignored. \\
9160   \c_@@_available_keys_str
9161 }
9162 {
9163   The~available~keys~are~(in~alphabetic~order):~
9164   b,~
9165   baseline,~
9166   c,~
9167   cell~space~bottom~limit,~
9168   cell~space~limits,~
9169   cell~space~top~limit,~
9170   code~after,~
9171   code~for~first~col,~
9172   code~for~first~row,~
9173   code~for~last~col,~
9174   code~for~last~row,~

```

```

9175 colortbl-like,~
9176 columns-width,~
9177 corners,~
9178 create-extra-nodes,~
9179 create-medium-nodes,~
9180 create-large-nodes,~
9181 extra-left-margin,~
9182 extra-right-margin,~
9183 first-col,~
9184 first-row,~
9185 hlines,~
9186 hvlines,~
9187 hvlines-except-borders,~
9188 last-col,~
9189 last-row,~
9190 left-margin,~
9191 light-syntax,~
9192 name,~
9193 nullify-dots,~
9194 pgf-node-code,~
9195 renew-dots,~
9196 respect-arraystretch,~
9197 right-margin,~
9198 rules~(with~the~subkeys~'color'~and~'width'),~
9199 small,~
9200 t,~
9201 vlines,~
9202 xdots/color,~
9203 xdots/shorten-start,~
9204 xdots/shorten-end,~
9205 xdots/shorten~and~
9206 xdots/line-style.
9207 }

```

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).

```

9208 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9209 {
9210   Unknown~key.\\
9211   The~key~'\l_keys_key_str'~is~unknown~for~the~
9212   \@@_full_name_env:. \\
9213   That~key~will~be~ignored. \\
9214   \c_@@_available_keys_str
9215 }
9216 {
9217   The~available~keys~are~(in~alphabetic~order):~
9218   b,~
9219   baseline,~
9220   c,~
9221   cell-space-bottom-limit,~
9222   cell-space-limits,~
9223   cell-space-top-limit,~
9224   code-after,~
9225   code-for-first-col,~
9226   code-for-first-row,~
9227   code-for-last-col,~
9228   code-for-last-row,~
9229   colortbl-like,~
9230   columns-type,~
9231   columns-width,~
9232   corners,~
9233   create-extra-nodes,~
9234   create-medium-nodes,~
9235   create-large-nodes,~

```

```

9236     extra-left-margin,~
9237     extra-right-margin,~
9238     first-col,~
9239     first-row,~
9240     hlines,~
9241     hvlines,~
9242     hvlines-except-borders,~
9243     l,~
9244     last-col,~
9245     last-row,~
9246     left-margin,~
9247     light-syntax,~
9248     name,~
9249     nullify-dots,~
9250     pgf-node-code,~
9251     r,~
9252     renew-dots,~
9253     respect-arraystretch,~
9254     right-margin,~
9255     rules~(with~the~subkeys~'color'~and~'width'),~
9256     small,~
9257     t,~
9258     vlines,~
9259     xdots/color,~
9260     xdots/shorten-start,~
9261     xdots/shorten-end,~
9262     xdots/shorten-and~
9263     xdots/line-style.
9264 }
9265 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9266 {
9267     Unknown~key.\\
9268     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9269     \{NiceTabular\}. \\
9270     That~key~will~be~ignored. \\
9271     \c_@@_available_keys_str
9272 }
9273 {
9274     The~available~keys~are~(in~alphabetic~order):~
9275     b,~
9276     baseline,~
9277     c,~
9278     caption,~
9279     cell-space-bottom-limit,~
9280     cell-space-limits,~
9281     cell-space-top-limit,~
9282     code-after,~
9283     code-for-first-col,~
9284     code-for-first-row,~
9285     code-for-last-col,~
9286     code-for-last-row,~
9287     colortbl-like,~
9288     columns-width,~
9289     corners,~
9290     custom-line,~
9291     create-extra-nodes,~
9292     create-medium-nodes,~
9293     create-large-nodes,~
9294     extra-left-margin,~
9295     extra-right-margin,~
9296     first-col,~
9297     first-row,~
9298     hlines,~

```

```

9299     hvlines,~
9300     hvlines-except-borders,~
9301     label,~
9302     last-col,~
9303     last-row,~
9304     left-margin,~
9305     light-syntax,~
9306     name,~
9307     notes~(several~subkeys),~
9308     nullify-dots,~
9309     pgf-node-code,~
9310     renew-dots,~
9311     respect-arraystretch,~
9312     right-margin,~
9313     rounded-corners,~
9314     rules~(with~the~subkeys~'color'~and~'width'),~
9315     short-caption,~
9316     t,~
9317     tabularnote,~
9318     vlines,~
9319     xdots/color,~
9320     xdots/shorten-start,~
9321     xdots/shorten-end,~
9322     xdots/shorten-and~
9323     xdots/line-style.
9324 }

9325 \@@_msg_new:nnn { Duplicate-name }
9326 {
9327     Duplicate-name.\
9328     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9329     the~same~environment~name~twice.~You~can~go~on,~but,~
9330     maybe,~you~will~have~incorrect~results~especially~
9331     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9332     message~again,~use~the~key~'allow-duplicate-names'~in~
9333     '\token_to_str:N \NiceMatrixOptions'.\
9334     \c_@@_available_keys_str
9335 }
9336 {
9337     The~names~already~defined~in~this~document~are:~
9338     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9339 }

9340 \@@_msg_new:nn { Option-auto-for-columns-width }
9341 {
9342     Erroneous~use.\
9343     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9344     That~key~will~be~ignored.
9345 }

```

# Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	17
6	Command for creation of rectangle nodes	22
7	The options	23
8	Important code used by <code>{NiceArrayWithDelims}</code>	33
9	The <code>\CodeBefore</code>	45
10	The environment <code>{NiceArrayWithDelims}</code>	49
11	We construct the preamble of the array	54
12	The redefinition of <code>\multicolumn</code>	68
13	The environment <code>{NiceMatrix}</code> and its variants	86
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	86
15	After the construction of the array	88
16	We draw the dotted lines	94
17	The actual instructions for drawing the dotted lines with <code>Tikz</code>	106
18	User commands available in the new environments	110
19	The command <code>\line</code> accessible in code-after	115
20	The command <code>\RowStyle</code>	117
21	Colors of cells, rows and columns	119
22	The vertical and horizontal rules	128
23	The key corners	143
24	The environment <code>{NiceMatrixBlock}</code>	145
25	The extra nodes	146
26	The blocks	151
27	How to draw the dotted lines transparently	168
28	Automatic arrays	169
29	The redefinition of the command <code>\dotfill</code>	170
30	The command <code>\diagbox</code>	170

31	The keyword <code>\CodeAfter</code>	172
32	The delimiters in the preamble	173
33	The command <code>\SubMatrix</code>	174
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	182
35	The command <code>\ShowCellNames</code>	185
36	We process the options at package loading	188
37	About the package underscore	189
38	Error messages of the package	190