

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

June 4, 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 6.20 of `nicematrix`, at the date of 2023/06/04.

```

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

```

19 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
20 {
21   \bool_if:NTF \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 curryfication.

```

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 redefine `\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_pgfutil@check@rerun \pgfutil@check@rerun

```

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

```

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_pgfutil@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`¹

```

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

```

¹See 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) correspondent 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”.²

```
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:

²We 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`.³
- 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 } } }

```

³More 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 parameter corresponds to the key `xdots/horizontal_labels`.

```

670 \bool_new:N \l_@@_xdots_h_labels_bool

```

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

```

671 \dim_new:N \l_@@_xdots_inter_dim
672 \hook_gput_code:nnn { begindocument } { . }
673 { \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).

```

674 \dim_new:N \l_@@_xdots_shorten_start_dim
675 \dim_new:N \l_@@_xdots_shorten_end_dim
676 \hook_gput_code:nnn { begindocument } { . }
677 {
678   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
679   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
680 }

```

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.

```

681 \dim_new:N \l_@@_xdots_radius_dim
682 \hook_gput_code:nnn { begindocument } { . }
683 { \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.

```

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

```

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

```

687 \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).

```

688 \tl_new:N \l_@@_baseline_tl
689 \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`).

```

690 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

691 \bool_new:N \l_@@_parallelize_diags_bool
692 \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.

```

693 \clist_new:N \l_@@_corners_clist

```

```

694 \dim_new:N \l_@@_notes_above_space_dim
695 \hook_gput_code:nnn { begindocument } { . }
696 { \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.).

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

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

```
698 \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).

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

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

```
700 \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.

```
701 \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”.

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

```
703 \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).

```
704 \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`).

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

```
706 \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`.

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

```
708 \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.

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

```
710 \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 “:”.

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

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

```
712 \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 function 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.

```

713 \bool_new:N \l_@@_delimiters_max_width_bool

714 \keys_define:nn { NiceMatrix / xdots }
715 {
716   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
717   horizontal-labels .default:n = true ,
718   line-style .code:n =
719   {
720     \bool_lazy_or:nnTF
721     { \cs_if_exist_p:N \tikzpicture }
722     { \str_if_eq_p:nn { #1 } { standard } }
723     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
724     { \@@_error:n { bad-option-for-line-style } }
725   } ,
726   line-style .value_required:n = true ,
727   color .tl_set:N = \l_@@_xdots_color_tl ,
728   color .value_required:n = true ,
729   shorten .code:n =
730   { \hook_gput_code:nnn { begindocument } { . }
731     {
732       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
733       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
734     } ,
735     shorten-start .code:n =
736     { \hook_gput_code:nnn { begindocument } { . }
737       { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } } ,
738     shorten-end .code:n =
739     { \hook_gput_code:nnn { begindocument } { . }
740       { \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.

```

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

```

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

```

752   down .tl_set:N = \l_@@_xdots_down_tl ,
753   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).

```

754   draw-first .code:n = \prg_do_nothing: ,
755   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
756 }

```

```

757 \keys_define:nn { NiceMatrix / rules }
758 {
759   color .tl_set:N = \l_@@_rules_color_tl ,
760   color .value_required:n = true ,
761   width .dim_set:N = \arrayrulewidth ,
762   width .value_required:n = true ,
763   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
764 }

```

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

```

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

```

```

817 {
818   \clist_set:Nn \l_@@_vlines_clist { all }
819   \clist_set:Nn \l_@@_hlines_clist { all }
820   \bool_set_true:N \l_@@_hvlines_bool
821   \bool_set_true:N \l_@@_except_borders_bool
822 } ,
823 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.

```

824 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
825 renew-dots .value_forbidden:n = true ,
826 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
827 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
828 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
829 create-extra-nodes .meta:n =
830   { create-medium-nodes , create-large-nodes } ,
831 left-margin .dim_set:N = \l_@@_left_margin_dim ,
832 left-margin .default:n = \arraycolsep ,
833 right-margin .dim_set:N = \l_@@_right_margin_dim ,
834 right-margin .default:n = \arraycolsep ,
835 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
836 margin .default:n = \arraycolsep ,
837 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
838 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
839 extra-margin .meta:n =
840   { extra-left-margin = #1 , extra-right-margin = #1 } ,
841 extra-margin .value_required:n = true ,
842 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
843 respect-arraystretch .default:n = true ,
844 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
845 pgf-node-code .value_required:n = true
846 }

```

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

```

847 \keys_define:nn { NiceMatrix / Env }
848 {
849   corners .clist_set:N = \l_@@_corners_clist ,
850   corners .default:n = { NW , SW , NE , SE } ,
851   code-before .code:n =
852     {
853       \tl_if_empty:nF { #1 }
854       {
855         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
856         \bool_set_true:N \l_@@_code_before_bool
857       }
858     } ,
859   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}`.

```

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

```

```

869 columns-width .value_required:n = true ,
870 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.

```

871 \legacy_if:nF { measuring@ }
872 {
873   \str_set:Nn \l_tmpa_str { #1 }
874   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
875   { \@@_error:nn { Duplicate-name } { #1 } }
876   { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
877   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
878 } ,
879 name .value_required:n = true ,
880 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
881 code-after .value_required:n = true ,
882 colortbl-like .code:n =
883   \bool_set_true:N \l_@@_colortbl_like_bool
884   \bool_set_true:N \l_@@_code_before_bool ,
885 colortbl-like .value_forbidden:n = true
886 }
887 \keys_define:nn { NiceMatrix / notes }
888 {
889   para .bool_set:N = \l_@@_notes_para_bool ,
890   para .default:n = true ,
891   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
892   code-before .value_required:n = true ,
893   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
894   code-after .value_required:n = true ,
895   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
896   bottomrule .default:n = true ,
897   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
898   style .value_required:n = true ,
899   label-in-tabular .code:n =
900     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
901   label-in-tabular .value_required:n = true ,
902   label-in-list .code:n =
903     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
904   label-in-list .value_required:n = true ,
905   enumitem-keys .code:n =
906     {
907       \hook_gput_code:nnn { begindocument } { . }
908       {
909         \IfPackageLoadedTF { enumitem }
910         { \setlist* [ tabularnotes ] { #1 } }
911         { }
912       }
913     } ,
914   enumitem-keys .value_required:n = true ,
915   enumitem-keys-para .code:n =
916     {
917       \hook_gput_code:nnn { begindocument } { . }
918       {
919         \IfPackageLoadedTF { enumitem }
920         { \setlist* [ tabularnotes* ] { #1 } }
921         { }
922       }
923     } ,
924   enumitem-keys-para .value_required:n = true ,
925   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
926   detect-duplicates .default:n = true ,
927   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
928 }

```

```

929 \keys_define:nn { NiceMatrix / delimiters }
930 {
931   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
932   max-width .default:n = true ,
933   color .tl_set:N = \l_@@_delimiters_color_tl ,
934   color .value_required:n = true ,
935 }

```

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

```

936 \keys_define:nn { NiceMatrix }
937 {
938   NiceMatrixOptions .inherit:n =
939     { NiceMatrix / Global } ,
940   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
941   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
942   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
943   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
944   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
945   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
946   CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
947   NiceMatrix .inherit:n =
948     {
949       NiceMatrix / Global ,
950       NiceMatrix / Env ,
951     } ,
952   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
953   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
954   NiceTabular .inherit:n =
955     {
956       NiceMatrix / Global ,
957       NiceMatrix / Env
958     } ,
959   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
960   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
961   NiceTabular / notes .inherit:n = NiceMatrix / notes ,
962   NiceArray .inherit:n =
963     {
964       NiceMatrix / Global ,
965       NiceMatrix / Env ,
966     } ,
967   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
968   NiceArray / rules .inherit:n = NiceMatrix / rules ,
969   pNiceArray .inherit:n =
970     {
971       NiceMatrix / Global ,
972       NiceMatrix / Env ,
973     } ,
974   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
975   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
976 }

```

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

```

977 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
978 {
979   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
980   delimiters / color .value_required:n = true ,
981   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
982   delimiters / max-width .default:n = true ,
983   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
984   delimiters .value_required:n = true ,

```

```

985 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
986 width .value_required:n = true ,
987 last-col .code:n =
988   \tl_if_empty:nF { #1 }
989     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
990     \int_zero:N \l_@@_last_col_int ,
991 small .bool_set:N = \l_@@_small_bool ,
992 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.

```

993 renew-matrix .code:n = \@@_renew_matrix: ,
994 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}`.

```

995 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.

```

996 columns-width .code:n =
997   \tl_if_eq:nnTF { #1 } { auto }
998     { \@@_error:n { Option-auto-for-columns-width } }
999     { \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.

```

1000 allow-duplicate-names .code:n =
1001   \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1002 allow-duplicate-names .value_forbidden:n = true ,
1003 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1004 notes .value_required:n = true ,
1005 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1006 sub-matrix .value_required:n = true ,
1007 matrix / columns-type .code:n =
1008   \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1009 matrix / columns-type .value_required:n = true ,
1010 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1011 caption-above .default:n = true ,
1012 unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1013 }

```

`\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.

```

1014 \NewDocumentCommand \NiceMatrixOptions { m }
1015 { \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.

```

1016 \keys_define:nn { NiceMatrix / NiceMatrix }
1017 {
1018   last-col .code:n = \tl_if_empty:nTF {#1}
1019     {
1020       \bool_set_true:N \l_@@_last_col_without_value_bool
1021       \int_set:Nn \l_@@_last_col_int { -1 }
1022     }
1023     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1024   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1025   columns-type .value_required:n = true ,

```

```

1026   l .meta:n = { columns-type = l } ,
1027   r .meta:n = { columns-type = r } ,
1028   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1029   delimiters / color .value_required:n = true ,
1030   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1031   delimiters / max-width .default:n = true ,
1032   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1033   delimiters .value_required:n = true ,
1034   small .bool_set:N = \l_@@_small_bool ,
1035   small .value_forbidden:n = true ,
1036   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1037 }

```

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

```

1038 \keys_define:nn { NiceMatrix / NiceArray }
1039 {

```

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.

```

1040   small .bool_set:N = \l_@@_small_bool ,
1041   small .value_forbidden:n = true ,
1042   last-col .code:n = \tl_if_empty:nF { #1 }
1043     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1044     \int_zero:N \l_@@_last_col_int ,
1045   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1046   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1047   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1048 }
1049 \keys_define:nn { NiceMatrix / pNiceArray }
1050 {
1051   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1052   last-col .code:n = \tl_if_empty:nF { #1 }
1053     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1054     \int_zero:N \l_@@_last_col_int ,
1055   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1056   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1057   delimiters / color .value_required:n = true ,
1058   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1059   delimiters / max-width .default:n = true ,
1060   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1061   delimiters .value_required:n = true ,
1062   small .bool_set:N = \l_@@_small_bool ,
1063   small .value_forbidden:n = true ,
1064   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1065   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1066   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1067 }

```

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

```

1068 \keys_define:nn { NiceMatrix / NiceTabular }
1069 {

```

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.

```

1070   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1071     \bool_set_true:N \l_@@_width_used_bool ,
1072   width .value_required:n = true ,
1073   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
1074   rounded-corners .default:n = 4 pt ,

```



```

1075 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1076 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1077 tabularnote .value_required:n = true ,
1078 caption .tl_set:N = \l_@@_caption_tl ,
1079 caption .value_required:n = true ,
1080 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1081 short-caption .value_required:n = true ,
1082 label .tl_set:N = \l_@@_label_tl ,
1083 label .value_required:n = true ,
1084 last-col .code:n = \tl_if_empty:nF {#1}
1085 { \@@_error:n { last-col~non-empty~for~NiceArray } }
1086 \int_zero:N \l_@@_last_col_int ,
1087 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1088 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1089 unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1090 }

```

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}`).

```

1091 \cs_new_protected:Npn \@@_cell_begin:w
1092 {

```

`\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.

```

1093 \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).

```

1094 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

1095 \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.

```

1096 \int_compare:nNnT \c@jCol = 1
1097 { \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).

```

1098 \hbox_set:Nw \l_@@_cell_box
1099 \bool_if:NF \l_@@_NiceTabular_bool
1100 {
1101   \c_math_toggle_token
1102   \bool_if:NT \l_@@_small_bool \scriptstyle
1103 }
1104 \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.

```

1105 \int_compare:nNnTF \c@iRow = 0
1106 {
1107   \int_compare:nNnT \c@jCol > 0
1108   {

```

```

1109         \l_@@_code_for_first_row_tl
1110         \xglobal \colorlet { nicematrix-first-row } { . }
1111     }
1112 }
1113 {
1114     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1115     {
1116         \l_@@_code_for_last_row_tl
1117         \xglobal \colorlet { nicematrix-last-row } { . }
1118     }
1119 }
1120 }

```

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.

```

1121 \cs_new_protected:Npn \@@_begin_of_row:
1122 {
1123     \int_gincr:N \c@iRow
1124     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1125     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1126     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1127     \pgfpicture
1128     \pgfrememberpicturepositiononpagetrue
1129     \pgfcoordinate
1130     { \@@_env: - row - \int_use:N \c@iRow - base }
1131     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1132     \str_if_empty:NF \l_@@_name_str
1133     {
1134         \pgfnodelalias
1135         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1136         { \@@_env: - row - \int_use:N \c@iRow - base }
1137     }
1138     \endpgfpicture
1139 }

```

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.

```

1140 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1141 {
1142     \int_compare:nNnTF \c@iRow = 0
1143     {
1144         \dim_gset:Nn \g_@@_dp_row_zero_dim
1145         { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1146         \dim_gset:Nn \g_@@_ht_row_zero_dim
1147         { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1148     }
1149     {
1150         \int_compare:nNnT \c@iRow = 1
1151         {
1152             \dim_gset:Nn \g_@@_ht_row_one_dim
1153             { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1154         }
1155     }
1156 }
1157 \cs_new_protected:Npn \@@_rotate_cell_box:
1158 {
1159     \box_rotate:Nn \l_@@_cell_box { 90 }

```

```

1160 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1161 {
1162     \vbox_set_top:Nn \l_@@_cell_box
1163     {
1164         \vbox_to_zero:n { }
1165         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1166         \box_use:N \l_@@_cell_box
1167     }
1168 }
1169 \bool_gset_false:N \g_@@_rotate_bool
1170 }
1171 \cs_new_protected:Npn \@@_adjust_size_box:
1172 {
1173     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1174     {
1175         \box_set_wd:Nn \l_@@_cell_box
1176         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1177         \dim_gzero:N \g_@@_blocks_wd_dim
1178     }
1179     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1180     {
1181         \box_set_dp:Nn \l_@@_cell_box
1182         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1183         \dim_gzero:N \g_@@_blocks_dp_dim
1184     }
1185     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1186     {
1187         \box_set_ht:Nn \l_@@_cell_box
1188         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1189         \dim_gzero:N \g_@@_blocks_ht_dim
1190     }
1191 }
1192 \cs_new_protected:Npn \@@_cell_end:
1193 {
1194     \@@_math_toggle_token:
1195     \hbox_set_end:
1196     \@@_cell_end_i:
1197 }
1198 \cs_new_protected:Npn \@@_cell_end_i:
1199 {

```

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.

```

1200 \g_@@_cell_after_hook_tl
1201 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1202 \@@_adjust_size_box:
1203 \box_set_ht:Nn \l_@@_cell_box
1204 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1205 \box_set_dp:Nn \l_@@_cell_box
1206 { \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”).

```

1207 \dim_gset:Nn \g_@@_max_cell_width_dim
1208 { \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”.

```

1209 \@@_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.

```

1210 \bool_if:NTF \g_@@_empty_cell_bool
1211 { \box_use_drop:N \l_@@_cell_box }
1212 {
1213   \bool_lazy_or:nnTF
1214     \g_@@_not_empty_cell_bool
1215     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1216     \@@_node_for_cell:
1217     { \box_use_drop:N \l_@@_cell_box }
1218 }
1219 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1220 \bool_gset_false:N \g_@@_empty_cell_bool
1221 \bool_gset_false:N \g_@@_not_empty_cell_bool
1222 }

```

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`).

```

1223 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1224 {
1225   \@@_math_toggle_token:
1226   \hbox_set_end:
1227   \bool_if:NF \g_@@_rotate_bool
1228   {
1229     \hbox_set:Nn \l_@@_cell_box
1230     {
1231       \makebox [ \l_@@_col_width_dim ] [ s ]
1232       { \hbox_unpack_drop:N \l_@@_cell_box }
1233     }
1234   }
1235   \@@_cell_end_i:
1236 }

```

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

```

1237 \pgfset
1238 {
1239   nicematrix / cell-node /.style =
1240   {
1241     inner~sep = \c_zero_dim ,
1242     minimum~width = \c_zero_dim
1243   }
1244 }
1245 \cs_new_protected:Npn \@@_node_for_cell:
1246 {

```

```

1247 \pgfpicture
1248 \pgfsetbaseline \c_zero_dim
1249 \pgfrememberpicturepositiononpagetrue
1250 \pgfset { nicematrix / cell-node }
1251 \pgfnode
1252 { rectangle }
1253 { base }
1254 {

```

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

```

1255 \set@color
1256 \box_use_drop:N \l_@@_cell_box
1257 }
1258 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1259 { \l_@@_pgf_node_code_tl }
1260 \str_if_empty:NF \l_@@_name_str
1261 {
1262 \pgfnodealias
1263 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1264 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1265 }
1266 \endpgfpicture
1267 }

```

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.

```

1268 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1269 {
1270 \cs_new_protected:Npn \@@_patch_node_for_cell:
1271 {
1272 \hbox_set:Nn \l_@@_cell_box
1273 {
1274 \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1275 \hbox_overlap_left:n
1276 {
1277 \pgfsys@markposition
1278 { \@@_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.

```

1279 #1
1280 }
1281 \box_use:N \l_@@_cell_box
1282 \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1283 \hbox_overlap_left:n
1284 {
1285 \pgfsys@markposition
1286 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1287 #1
1288 }
1289 }
1290 }
1291 }

```

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

```

1292 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1293 {
1294 \@@_patch_node_for_cell:n
1295 { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1296 }
1297 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

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

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

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

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

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

```
1298 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1299 {
1300   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1301   { \g_@@_#2 _ lines _ tl }
1302   {
1303     \use:c { @@ _ draw _ #2 : nnn }
1304     { \int_use:N \c@iRow }
1305     { \int_use:N \c@jCol }
1306     { \exp_not:n { #3 } }
1307   }
1308 }

1309 \cs_new_protected:Npn \@@_array:n
1310 {
1311   \bool_if:NTF \l_@@_NiceTabular_bool
1312   { \dim_set_eq:NN \col@sep \tabcolsep }
1313   { \dim_set_eq:NN \col@sep \arraycolsep }
1314   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1315   { \cs_set_nopar:Npn \@halignto { } }
1316   { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1317   \@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.

1318   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1319 }
1320 \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.

```
1321 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1322 \cs_new_protected:Npn \@@_create_row_node:
1323 {
1324   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1325   {
1326     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1327     \@@_create_row_node_i:
1328   }
1329 }
```

```

1330 \cs_new_protected:Npn \@@_create_row_node_i:
1331 {

```

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

```

1332   \hbox
1333   {
1334     \bool_if:NT \l_@@_code_before_bool
1335     {
1336       \vtop
1337       {
1338         \skip_vertical:N 0.5\arrayrulewidth
1339         \pgfsys@markposition
1340         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1341         \skip_vertical:N -0.5\arrayrulewidth
1342       }
1343     }
1344     \pgfpicture
1345     \pgfrememberpicturepositiononpagetrue
1346     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1347     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1348     \str_if_empty:NF \l_@@_name_str
1349     {
1350       \pgfnodealias
1351       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1352       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1353     }
1354     \endpgfpicture
1355   }
1356 }

```

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

```

1357 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1358 \cs_new_protected:Npn \@@_everycr_i:
1359 {
1360   \int_gzero:N \c@jCol
1361   \bool_gset_false:N \g_@@_after_col_zero_bool
1362   \bool_if:NF \g_@@_row_of_col_done_bool
1363   {
1364     \@@_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).

```

1365     \tl_if_empty:NF \l_@@_hlines_clist
1366     {
1367       \tl_if_eq:NnF \l_@@_hlines_clist { all }
1368       {
1369         \exp_args:NNx
1370         \clist_if_in:NnT
1371         \l_@@_hlines_clist
1372         { \int_eval:n { \c@iRow + 1 } }
1373       }
1374     {

```

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.

```

1375       \int_compare:nNnT \c@iRow > { -1 }
1376       {
1377         \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@`.

```

1378         { \hrule height \arrayrulewidth width \c_zero_dim }

```

```

1379         }
1380     }
1381 }
1382 }
1383 }

```

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`).

```

1384 \cs_set_protected:Npn \@@_newcolumntype #1
1385 {
1386     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1387     \peek_meaning:NTF [
1388         { \newcol@ #1 }
1389         { \newcol@ #1 [ 0 ] }
1390     }

```

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

```

1391 \cs_set_protected:Npn \@@_renew_dots:
1392 {
1393     \cs_set_eq:NN \ldots \@@_Ldots
1394     \cs_set_eq:NN \cdots \@@_Cdots
1395     \cs_set_eq:NN \vdots \@@_Vdots
1396     \cs_set_eq:NN \ddots \@@_Ddots
1397     \cs_set_eq:NN \iddots \@@_Iddots
1398     \cs_set_eq:NN \dots \@@_Ldots
1399     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1400 }

```

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

```

1401 \cs_new_protected:Npn \@@_colortbl_like:
1402 {
1403     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1404     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1405     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1406 }

```

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

```

1407 \cs_new_protected:Npn \@@_pre_array_ii:
1408 {

```

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

```

1409     \int_gzero:N \g_@@_total_X_weight_int
1410     \@@_expand_clist:N \l_@@_hlines_clist
1411     \@@_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 ⁴.

```

1412     \IfPackageLoadedTF { booktabs }
1413     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1414     { }
1415     \box_clear_new:N \l_@@_cell_box
1416     \normalbaselines

```

⁴cf. `\nicematrix@redefine@check@rerun`

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}`).

```

1417 \bool_if:NT \l_@@_small_bool
1418 {
1419     \cs_set_nopar:Npn \arraystretch { 0.47 }
1420     \dim_set:Nn \arraycolsep { 1.45 pt }
1421 }

1422 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1423 {
1424     \tl_put_right:Nn \@@_begin_of_row:
1425     {
1426         \pgfsys@markposition
1427         { \@@_env: - row - \int_use:N \c@iRow - base }
1428     }
1429 }

```

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`.

```

1430 \cs_set_nopar:Npn \ialign
1431 {
1432     \IfPackageLoadedTF { colortbl }
1433     {
1434         \CT@everycr
1435         {
1436             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1437             \@@_everycr:
1438         }
1439     }
1440     { \everycr { \@@_everycr: } }
1441     \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`⁵ 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`.

```

1442 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1443 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1444 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1445 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1446 \dim_gzero_new:N \g_@@_ht_row_one_dim
1447 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1448 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1449 \dim_gzero_new:N \g_@@_ht_last_row_dim
1450 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1451 \dim_gzero_new:N \g_@@_dp_last_row_dim
1452 \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`.

```

1453 \cs_set_eq:NN \ialign \@@_old_ialign:
1454 \halign
1455 }

```

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

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).

```

1456 \cs_set_eq:NN \@@_old_ldots \ldots
1457 \cs_set_eq:NN \@@_old_cdots \cdots
1458 \cs_set_eq:NN \@@_old_vdots \vdots
1459 \cs_set_eq:NN \@@_old_ddots \ddots
1460 \cs_set_eq:NN \@@_old_iddots \iddots
1461 \bool_if:NTF \l_@@_standard_cline_bool
1462 { \cs_set_eq:NN \cline \@@_standard_cline }
1463 { \cs_set_eq:NN \cline \@@_cline }
1464 \cs_set_eq:NN \Ldots \@@_Ldots
1465 \cs_set_eq:NN \Cdots \@@_Cdots
1466 \cs_set_eq:NN \Vdots \@@_Vdots
1467 \cs_set_eq:NN \Ddots \@@_Ddots
1468 \cs_set_eq:NN \Iddots \@@_Iddots
1469 \cs_set_eq:NN \Hline \@@_Hline:
1470 \cs_set_eq:NN \Hspace \@@_Hspace:
1471 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1472 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1473 \cs_set_eq:NN \Block \@@_Block:
1474 \cs_set_eq:NN \rotate \@@_rotate:
1475 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1476 \cs_set_eq:NN \dotfill \@@_dotfill:
1477 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1478 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1479 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1480 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1481 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1482 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1483 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1484 \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.

```

1485 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1486 \hook_gput_code:nnn { env / tabular / begin } { . }
1487 { \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!).

```

1488 \tl_if_exist:NT \l_@@_note_in_caption_tl
1489 {
1490   \tl_if_empty:NF \l_@@_note_in_caption_tl
1491   {
1492     \int_gset_eq:NN \g_@@_notes_caption_int
1493     { \l_@@_note_in_caption_tl }
1494     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1495   }
1496 }

```

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).

```

1497 \seq_gclear:N \g_@@_multicolumn_cells_seq
1498 \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).

```
1499 \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).

```
1500 \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.

```
1501 \int_gzero_new:N \g_@@_col_total_int
```

```
1502 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

```
1503 \@@_renew_NC@rewrite@S:
```

```
1504 \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.

```
1505 \tl_gclear_new:N \g_@@_Cdots_lines_tl
```

```
1506 \tl_gclear_new:N \g_@@_Ldots_lines_tl
```

```
1507 \tl_gclear_new:N \g_@@_Vdots_lines_tl
```

```
1508 \tl_gclear_new:N \g_@@_Ddots_lines_tl
```

```
1509 \tl_gclear_new:N \g_@@_Iddots_lines_tl
```

```
1510 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl
```

```
1511 \tl_gclear:N \g_nicematrix_code_before_tl
```

```
1512 \tl_gclear:N \g_@@_pre_code_before_tl
```

```
1513 }
```

This is the end of \@@_pre_array_ii:.

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

```
1514 \cs_new_protected:Npn \@@_pre_array:
```

```
1515 {
```

```
1516 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
```

```
1517 \int_gzero_new:N \c@iRow
```

```
1518 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
```

```
1519 \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).

```
1520 \int_compare:nNnT \l_@@_last_row_int = { -1 }
```

```
1521 {
```

```
1522 \bool_set_true:N \l_@@_last_row_without_value_bool
```

```
1523 \bool_if:NT \g_@@_aux_found_bool
```

```
1524 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
```

```
1525 }
```

```
1526 \int_compare:nNnT \l_@@_last_col_int = { -1 }
```

```
1527 {
```

```
1528 \bool_if:NT \g_@@_aux_found_bool
```

```
1529 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
```

```
1530 }
```

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”.

```

1531 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1532 {
1533   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1534   {
1535     \dim_gset:Nn \g_@@_ht_last_row_dim
1536     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1537     \dim_gset:Nn \g_@@_dp_last_row_dim
1538     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1539   }
1540 }

1541 \seq_gclear:N \g_@@_cols_vlism_seq
1542 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1543 \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.

```

1544 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1545 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1546 \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.

```

1547 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1548 \@@_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.

```

1549 \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]`).

```

1550 \dim_zero_new:N \l_@@_left_delim_dim
1551 \dim_zero_new:N \l_@@_right_delim_dim
1552 \bool_if:NTF \g_@@_NiceArray_bool
1553 {
1554   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1555   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1556 }
1557 {

```

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

```

1558 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1559 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1560 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1561 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1562 }

```

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).

```

1563   \hbox_set:Nw \l_@@_the_array_box
1564   \skip_horizontal:N \l_@@_left_margin_dim
1565   \skip_horizontal:N \l_@@_extra_left_margin_dim
1566   \c_math_toggle_token
1567   \bool_if:NTF \l_@@_light_syntax_bool
1568     { \use:c { @@-light-syntax } }
1569     { \use:c { @@-normal-syntax } }
1570 }

```

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

```

1571 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1572 {
1573   \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1574   \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...

```

1575   \@@_pre_array:
1576 }

```

9 The `\CodeBefore`

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

```

1577 \cs_new_protected:Npn \@@_pre_code_before:
1578 {

```

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).

```

1579   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1580   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1581   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1582   \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.

```

1583   \pgfsys@markposition { \@@_env: - position }
1584   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1585   \pgfpicture
1586   \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1587   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1588   {
1589     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1590     \pgfcoordinate { \@@_env: - row - ##1 }
1591     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1592   }

```

Now, the recreation of the col nodes.

```

1593 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1594 {
1595     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1596     \pgfcoordinate { \@@_env: - col - ##1 }
1597     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1598 }

```

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

```

1599 \@@_create_diag_nodes:

```

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

```

1600 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1601 \endpgfpicture

```

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

```

1602 \@@_create_blocks_nodes:
1603 \IfPackageLoadedTF { tikz }
1604 {
1605     \tikzset
1606     {
1607         every~picture / .style =
1608         { overlay , name~prefix = \@@_env: - }
1609     }
1610 }
1611 { }
1612 \cs_set_eq:NN \cellcolor \@@_cellcolor
1613 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1614 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1615 \cs_set_eq:NN \rowcolor \@@_rowcolor
1616 \cs_set_eq:NN \rowcolors \@@_rowcolors
1617 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1618 \cs_set_eq:NN \arraycolor \@@_arraycolor
1619 \cs_set_eq:NN \columncolor \@@_columncolor
1620 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1621 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1622 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1623 }

1624 \cs_new_protected:Npn \@@_exec_code_before:
1625 {
1626     \seq_gclear_new:N \g_@@_colors_seq
1627     \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1628     \group_begin:

```

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

```

1629 \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`).

```

1630 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1631 {
1632     \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1633     \@@_rescan_for_spanish:N \l_@@_code_before_tl
1634 }

```

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`.

```
1635 \exp_last_unbraced:NV \@@_CodeBefore_keys:
1636 \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.

```
1637 \@@_actually_color:
1638 \l_@@_code_before_tl
1639 \q_stop
1640 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1641 \group_end:
1642 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1643 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1644 }
```

```
1645 \keys_define:nn { NiceMatrix / CodeBefore }
1646 {
1647   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1648   create-cell-nodes .default:n = true ,
1649   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1650   sub-matrix .value_required:n = true ,
1651   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1652   delimiters / color .value_required:n = true ,
1653   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1654 }
1655 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1656 {
1657   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1658   \@@_CodeBefore:w
1659 }
```

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.

```
1660 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1661 {
1662   \bool_if:NT \g_@@_aux_found_bool
1663   {
1664     \@@_pre_code_before:
1665     #1
1666   }
1667 }
```

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.

```
1668 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1669 {
1670   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1671   {
1672     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1673     \pgfcoordinate { \@@_env: - row - ##1 - base }
1674     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1675     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
```

```

1676 {
1677   \cs_if_exist:cT
1678   { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1679   {
1680     \pgfsys@getposition
1681     { \@@_env: - ##1 - #####1 - NW }
1682     \@@_node_position:
1683     \pgfsys@getposition
1684     { \@@_env: - ##1 - #####1 - SE }
1685     \@@_node_position_i:
1686     \@@_pgf_rect_node:nnn
1687     { \@@_env: - ##1 - #####1 }
1688     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1689     { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1690   }
1691 }
1692 }
1693 \int_step_inline:nn \c@iRow
1694 {
1695   \pgfnodealias
1696   { \@@_env: - ##1 - last }
1697   { \@@_env: - ##1 - \int_use:N \c@jCol }
1698 }
1699 \int_step_inline:nn \c@jCol
1700 {
1701   \pgfnodealias
1702   { \@@_env: - last - ##1 }
1703   { \@@_env: - \int_use:N \c@iRow - ##1 }
1704 }
1705 \@@_create_extra_nodes:
1706 }

1707 \cs_new_protected:Npn \@@_create_blocks_nodes:
1708 {
1709   \pgfpicture
1710   \pgf@relevantforpicturesizefalse
1711   \pgfrememberpicturepositiononpagetrue
1712   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1713   { \@@_create_one_block_node:nnnnn ##1 }
1714   \endpgfpicture
1715 }

```

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.⁶

```

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

```

⁶Moreover, 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).


```

1731         { \dim_use:N \l_tmpb_dim }
1732         { \dim_use:N \l_@@_tmpc_dim }
1733         { \dim_use:N \l_@@_tmpd_dim }
1734     }
1735 }

1736 \cs_new_protected:Npn \@@_patch_for_revtext:
1737 {
1738     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1739     \cs_set_eq:NN \insert@column \insert@column@array
1740     \cs_set_eq:NN \@classx \@classx@array
1741     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1742     \cs_set_eq:NN \@arraycr \@arraycr@array
1743     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1744     \cs_set_eq:NN \array \array@array
1745     \cs_set_eq:NN \@array \@array@array
1746     \cs_set_eq:NN \@tabular \@tabular@array
1747     \cs_set_eq:NN \@mkpream \@mkpream@array
1748     \cs_set_eq:NN \endarray \endarray@array
1749     \cs_set:Npn \@tabarray { \@ifnextchar [ { \array } { \array [ c ] } }
1750     \cs_set:Npn \endtabular { \endarray $\egroup} % $
1751 }

```

10 The environment `{NiceArrayWithDelims}`

```

1752 \NewDocumentEnvironment { NiceArrayWithDelims }
1753 { m m O { } m ! O { } t \CodeBefore }
1754 {
1755     \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1756     \@@_provide_pgfsyspdfmark:
1757     \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.

```

1758     \bgroup

1759     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1760     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1761     \tl_gset:Nn \g_@@_preamble_tl { #4 }

1762     \int_gzero:N \g_@@_block_box_int
1763     \dim_zero:N \g_@@_width_last_col_dim
1764     \dim_zero:N \g_@@_width_first_col_dim
1765     \bool_gset_false:N \g_@@_row_of_col_done_bool
1766     \str_if_empty:NT \g_@@_name_env_str
1767     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1768     \bool_if:NTF \l_@@_NiceTabular_bool
1769     \mode_leave_vertical:
1770     \@@_test_if_math_mode:
1771     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1772     \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁷. 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

⁷e.g. `\color[rgb]{0.5,0.5,0}`

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.

```
1773 \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.

```
1774 \cs_if_exist:NT \tikz@library@external@loaded
1775 {
1776   \tikzexternaldisable
1777   \cs_if_exist:NT \ifstandalone
1778     { \tikzset { external / optimize = false } }
1779 }
```

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

```
1780 \int_gincr:N \g_@@_env_int
1781 \bool_if:NF \l_@@_block_auto_columns_width_bool
1782 { \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`).

```
1783 \seq_gclear:N \g_@@_blocks_seq
1784 \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`.

```
1785 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1786 \seq_gclear:N \g_@@_pos_of_xdots_seq
1787 \tl_gclear_new:N \g_@@_code_before_tl
1788 \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.

```
1789 \bool_gset_false:N \g_@@_aux_found_bool
1790 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1791 {
1792   \bool_gset_true:N \g_@@_aux_found_bool
1793   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1794 }
```

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.

```
1795 \tl_gclear:N \g_@@_aux_tl
1796 \tl_if_empty:NF \g_@@_code_before_tl
1797 {
1798   \bool_set_true:N \l_@@_code_before_bool
1799   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1800 }
1801 \tl_if_empty:NF \g_@@_pre_code_before_tl
1802 { \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`.

```
1803 \bool_if:NTF \g_@@_NiceArray_bool
1804 { \keys_set:nn { NiceMatrix / NiceArray } }
1805 { \keys_set:nn { NiceMatrix / pNiceArray } }
1806 { #3 , #5 }

1807 \@@_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:`.

```
1808   \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1809 }
```

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

```
1810 {
1811   \bool_if:NTF \l_@@_light_syntax_bool
1812     { \use:c { end @@-light-syntax } }
1813     { \use:c { end @@-normal-syntax } }
1814   \c_math_toggle_token
1815   \skip_horizontal:N \l_@@_right_margin_dim
1816   \skip_horizontal:N \l_@@_extra_right_margin_dim
1817   \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.

```
1818   \bool_if:NT \l_@@_width_used_bool
1819   {
1820     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1821       { \@@_error_or_warning:n { width-without~X-columns } }
1822   }
```

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 .

```
1823   \int_compare:nNnT \g_@@_total_X_weight_int > 0
1824   {
1825     \tl_gput_right:Nx \g_@@_aux_tl
1826     {
1827       \bool_set_true:N \l_@@_X_columns_aux_bool
1828       \dim_set:Nn \l_@@_X_columns_dim
1829       {
1830         \dim_compare:nNnTF
1831           {
1832             \dim_abs:n
1833               { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1834           }
1835           <
1836           { 0.001 pt }
1837           { \dim_use:N \l_@@_X_columns_dim }
1838           {
1839             \dim_eval:n
1840             {
1841               ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1842               / \int_use:N \g_@@_total_X_weight_int
1843               + \l_@@_X_columns_dim
1844             }
1845           }
1846       }
1847     }
1848   }
```

If the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the actual number of rows of the array).

```
1849   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1850   {
1851     \bool_if:NF \l_@@_last_row_without_value_bool
1852     {
1853       \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1854       {
```

```

1855         \@@_error:n { Wrong~last~row }
1856         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1857     }
1858 }
1859 }

```

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”.⁸

```

1860     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1861     \bool_if:nTF \g_@@_last_col_found_bool
1862     { \int_gdecr:N \c@jCol }
1863     {
1864         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1865         { \@@_error:n { last~col~not~used } }
1866     }

```

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

```

1867     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1868     \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).

```

1869     \int_compare:nNnT \l_@@_first_col_int = 0
1870     {
1871         \skip_horizontal:N \col@sep
1872         \skip_horizontal:N \g_@@_width_first_col_dim
1873     }

```

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.

```

1874     \bool_if:nTF \g_@@_NiceArray_bool
1875     {
1876         \str_case:VnF \l_@@_baseline_tl
1877         {
1878             b \@@_use_arraybox_with_notes_b:
1879             c \@@_use_arraybox_with_notes_c:
1880         }
1881         \@@_use_arraybox_with_notes:
1882     }

```

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).

```

1883     {
1884         \int_compare:nNnTF \l_@@_first_row_int = 0
1885         {
1886             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1887             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1888         }
1889         { \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”.⁹

```

1890     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1891     {
1892         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1893         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1894     }
1895     { \dim_zero:N \l_tmpb_dim }

```

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

⁹A 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).

```

1896     \hbox_set:Nn \l_tmpa_box
1897     {
1898         \c_math_toggle_token
1899         \@@_color:V \l_@@_delimiters_color_tl
1900         \exp_after:wN \left \g_@@_left_delim_tl
1901         \vcenter
1902         {

```

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.

```

1903         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1904         \hbox
1905         {
1906             \bool_if:NTF \l_@@_NiceTabular_bool
1907             { \skip_horizontal:N -\tabcolsep }
1908             { \skip_horizontal:N -\arraycolsep }
1909             \@@_use_arraybox_with_notes_c:
1910             \bool_if:NTF \l_@@_NiceTabular_bool
1911             { \skip_horizontal:N -\tabcolsep }
1912             { \skip_horizontal:N -\arraycolsep }
1913         }

```

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

```

1914         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1915     }

```

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.

```

1916         \@@_color:V \l_@@_delimiters_color_tl
1917         \exp_after:wN \right \g_@@_right_delim_tl
1918         \c_math_toggle_token
1919     }

```

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.

```

1920     \bool_if:NTF \l_@@_delimiters_max_width_bool
1921     {
1922         \@@_put_box_in_flow_bis:nn
1923         \g_@@_left_delim_tl \g_@@_right_delim_tl
1924     }
1925     \@@_put_box_in_flow:
1926 }

```

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).

```

1927     \bool_if:NT \g_@@_last_col_found_bool
1928     {
1929         \skip_horizontal:N \g_@@_width_last_col_dim
1930         \skip_horizontal:N \col@sep
1931     }
1932     \bool_if:NF \l_@@_Matrix_bool
1933     {
1934         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1935         { \@@_warning_gredirect_none:n { columns-not-used } }
1936     }
1937     \@@_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.

```

1938     \egroup

```

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

```

1939     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1940     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1941     \iow_now:Nx \@mainaux

```

```

1942     {
1943       \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1944       { \exp_not:V \g_@@_aux_tl }
1945     }
1946     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1947     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1948   }

```

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.¹⁰

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

```

1949 \cs_new_protected:Npn \@@_transform_preamble:
1950 {

```

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.

```

1951   \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).

```

1952   \bool_if:NF \l_@@_Matrix_bool
1953   {
1954     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1955     \@@_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).

```

1956     \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).

```

1957     \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1958     \@tempswatruw

```

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

```

1959     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

¹⁰Be 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.

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`.

```

1960      \int_gzero:N \c@jCol
1961      \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1962      \bool_gset_false:N \g_tmpb_bool
1963      \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1964      {
1965          \tl_gset:Nn \g_@@_preamble_tl
1966              { ! { \skip_horizontal:N \arrayrulewidth } }
1967      }
1968      {
1969          \clist_if_in:NnT \l_@@_vlines_clist 1
1970          {
1971              \tl_gset:Nn \g_@@_preamble_tl
1972                  { ! { \skip_horizontal:N \arrayrulewidth } }
1973          }
1974      }

```

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`).

```

1975      \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1976      \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

1977      \int_zero:N \l_tmpa_int

```

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

```

1978      \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1979      \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1980      }

```

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

```

1981      \bool_if:NT \l_@@_colortbl_like_bool
1982      {
1983          \regex_replace_all:NnN
1984              \c_@@_columncolor_regex
1985              { \c { @@_columncolor_preamble } }
1986              \g_@@_preamble_tl
1987      }

```

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

```

1988      \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}`.

```

1989      \bool_lazy_or:nnT
1990          { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1991          { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1992          { \bool_gset_false:N \g_@@_NiceArray_bool }

```

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

```

1993      \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).

```

1994 \int_compare:nNnTF \l_@@_first_col_int = 0
1995 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1996 {
1997   \bool_lazy_all:nT
1998   {
1999     \g_@@_NiceArray_bool
2000     { \bool_not_p:n \l_@@_NiceTabular_bool }
2001     { \tl_if_empty_p:N \l_@@_vlines_clist }
2002     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2003   }
2004   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
2005 }
2006 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2007 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
2008 {
2009   \bool_lazy_all:nT
2010   {
2011     \g_@@_NiceArray_bool
2012     { \bool_not_p:n \l_@@_NiceTabular_bool }
2013     { \tl_if_empty_p:N \l_@@_vlines_clist }
2014     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2015   }
2016   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
2017 }

```

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`).

```

2018 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2019 {
2020   \tl_gput_right:Nn \g_@@_preamble_tl
2021   { > { \@@_error_too_much_cols: } 1 }
2022 }
2023 }

```

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

```

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

```



```

2049     \}      { \@@_patch_preamble_ix:nn #1 }
2050     \right { \@@_patch_preamble_ix:nn }
2051     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}`.

```

2052     \@@_X   { \@@_patch_preamble_x:n }
2053     \q_stop { }
2054   }
2055   {
2056     \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2057     {
2058       \seq_gput_right:Nx \g_@@_cols_vlism_seq
2059       { \int_eval:n { \c@jCol + 1 } }
2060       \tl_gput_right:Nx \g_@@_preamble_tl
2061       { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2062       \@@_patch_preamble:n
2063     }

```

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).

```

2064     {
2065       \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2066       {
2067         \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2068         \@@_patch_preamble:n
2069       }
2070       {
2071         \tl_if_eq:nnT { #1 } { S }
2072         { \@@_fatal:n { unknown~column~type~S } }
2073         { \@@_fatal:nn { unknown~column~type } { #1 } }
2074       }
2075     }
2076   }
2077 }

```

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`

```

2078 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2079 {
2080   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2081   \tl_gclear:N \g_@@_pre_cell_tl
2082   \tl_gput_right:Nn \g_@@_preamble_tl
2083   {
2084     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2085     #1
2086     < \@@_cell_end:
2087   }

```

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

```

2088     \int_gincr:N \c@jCol
2089     \@@_patch_preamble_xi:n
2090   }

```

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

```

2091 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2092 {

```

```

2093 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2094 \@@_patch_preamble:n
2095 }

```

For |

```

2096 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2097 {
\l_tmpa_int is the number of successive occurrences of |
2098 \int_incr:N \l_tmpa_int
2099 \@@_patch_preamble_iii_i:n
2100 }
2101 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2102 {
2103 \str_if_eq:nnTF { #1 } |
2104 { \@@_patch_preamble_iii:n | }
2105 {
2106 \dim_set:Nn \l_tmpa_dim
2107 {
2108 \arrayrulewidth * \l_tmpa_int
2109 + \doublerulesep * ( \l_tmpa_int - 1 )
2110 }
2111 \tl_gput_right:Nx \g_@@_preamble_tl
2112 {

```

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

```

2113 \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_tmpa_dim } } }
2114 }
2115 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2116 {
2117 \@@_vline:n
2118 {
2119 position = \int_eval:n { \c@jCol + 1 } ,
2120 multiplicity = \int_use:N \l_tmpa_int ,
2121 total-width = \dim_use:N \l_tmpa_dim % added 2022-08-06
2122 }

```

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.

```

2123 }
2124 \int_zero:N \l_tmpa_int
2125 \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2126 \@@_patch_preamble:n #1
2127 }
2128 }
2129 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2130 {
2131 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2132 \@@_patch_preamble:n
2133 }
2134 \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.

```

2135 \keys_define:nn { WithArrows / p-column }
2136 {
2137 r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2138 r .value_forbidden:n = true ,
2139 c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2140 c .value_forbidden:n = true ,
2141 l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2142 l .value_forbidden:n = true ,

```

```

2143 R .code:n =
2144     \IfPackageLoadedTF { ragged2e }
2145     { \str_set:Nn \l_@@_hpos_col_str { R } }
2146     {
2147         \@@_error_or_warning:n { ragged2e~not~loaded }
2148         \str_set:Nn \l_@@_hpos_col_str { r }
2149     } ,
2150 R .value_forbidden:n = true ,
2151 L .code:n =
2152     \IfPackageLoadedTF { ragged2e }
2153     { \str_set:Nn \l_@@_hpos_col_str { L } }
2154     {
2155         \@@_error_or_warning:n { ragged2e~not~loaded }
2156         \str_set:Nn \l_@@_hpos_col_str { l }
2157     } ,
2158 L .value_forbidden:n = true ,
2159 C .code:n =
2160     \IfPackageLoadedTF { ragged2e }
2161     { \str_set:Nn \l_@@_hpos_col_str { C } }
2162     {
2163         \@@_error_or_warning:n { ragged2e~not~loaded }
2164         \str_set:Nn \l_@@_hpos_col_str { c }
2165     } ,
2166 C .value_forbidden:n = true ,
2167 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2168 S .value_forbidden:n = true ,
2169 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2170 p .value_forbidden:n = true ,
2171 t .meta:n = p ,
2172 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2173 m .value_forbidden:n = true ,
2174 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2175 b .value_forbidden:n = true ,
2176 }

```

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

```

2177 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2178 {
2179     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2180     \@@_patch_preamble_iv_i:n
2181 }

2182 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2183 {
2184     \str_if_eq:nnTF { #1 } { [ ]
2185         { \@@_patch_preamble_iv_ii:w [ ]
2186             { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2187         }
2188     \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2189     { \@@_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.

```

2190 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2191 {

```

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).

```

2192     \str_set:Nn \l_@@_hpos_col_str { j }
2193     \tl_set:Nn \l_tmpa_tl { #1 }
2194     \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2195     \@@_keys_p_column:V \l_tmpa_tl

```

```

2196 \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2197 }
2198 \cs_new_protected:Npn \@@_keys_p_column:n #1
2199 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2200 \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`.

```

2201 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2202 {
2203   \use:x
2204   {
2205     \@@_patch_preamble_iv_v:nnnnnnnn
2206     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2207     { \dim_eval:n { #1 } }
2208     {

```

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.

```

2209       \str_if_eq:VnTF \l_@@_hpos_col_str j
2210       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
2211       {
2212         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2213         { \str_lowercase:V \l_@@_hpos_col_str }
2214       }
2215     \str_case:Vn \l_@@_hpos_col_str
2216     {
2217       c { \exp_not:N \centering }
2218       l { \exp_not:N \raggedright }
2219       r { \exp_not:N \raggedleft }
2220       C { \exp_not:N \Centering }
2221       L { \exp_not:N \RaggedRight }
2222       R { \exp_not:N \RaggedLeft }
2223     }
2224   }
2225   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2226   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2227   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2228   { #2 }
2229   {
2230     \str_case:VnF \l_@@_hpos_col_str
2231     {
2232       { j } { c }
2233       { si } { c }
2234     }

```

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

```

2235       { \str_lowercase:V \l_@@_hpos_col_str }
2236     }
2237   }

```

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

```

2238   \int_gincr:N \c@jCol
2239   \@@_patch_preamble_xi:n
2240 }

```

#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*.

```

2241 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2242 {
2243   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2244     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2245     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2246   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2247   \tl_gclear:N \g_@@_pre_cell_tl
2248   \tl_gput_right:Nn \g_@@_preamble_tl
2249     {
2250       > {

```

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.

```

2251       \dim_set:Nn \l_@@_col_width_dim { #2 }
2252       \@@_cell_begin:w
2253       \begin { #7 } [ #1 ] { #2 }

```

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

```

2254       \everypar
2255       {
2256         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2257         \everypar { }
2258       }

```

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

```

2259       #3

```

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

```

2260       \g_@@_row_style_tl
2261       \arraybackslash
2262       #5
2263     }
2264     #8
2265     < {
2266       #6

```

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

```

2267       \@finalstrut \@arstrutbox
2268       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
2269       \end { #7 }

```

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

```

2270       #4
2271       \@@_cell_end:
2272     }
2273   }
2274 }

```

```

2275 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2276 {
2277   \peek_meaning:NT \unskip
2278   {
2279     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2280     {
2281       \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.

```

2282         \skip_horizontal:N \l_@@_col_width_dim
2283     }
2284 }
2285 #1
2286 }

2287 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2288 {
2289     \peek_meaning:NT \__siunitx_table_skip:n
2290     {
2291         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2292         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2293     }
2294     #1
2295 }
```

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.

```

2296 \cs_new_protected:Npn \@@_center_cell_box:
2297 {
```

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

```

2298     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2299     {
2300         \int_compare:nNnT
2301         { \box_ht:N \l_@@_cell_box }
2302         >
```

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).

```

2303         { \box_ht:N \strutbox }
2304     {
2305         \hbox_set:Nn \l_@@_cell_box
2306         {
2307             \box_move_down:nn
2308             {
2309                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2310                 + \baselineskip ) / 2
2311             }
2312             { \box_use:N \l_@@_cell_box }
2313         }
2314     }
2315 }
2316 }
```

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

```

2317 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2318 {
2319     \str_if_eq:nnTF { #1 } { [ ] }
2320     { \@@_patch_preamble_v_i:w [ ]
2321     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2322     }
2323 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2324 { \@@_patch_preamble_v_ii:nn { #1 } }
2325 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2326 {
```

```

2327 \str_set:Nn \l_@@_vpos_col_str { p }
2328 \str_set:Nn \l_@@_hpos_col_str { j }
2329 \tl_set:Nn \l_tmpa_tl { #1 }
2330 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2331 \@@_keys_p_column:V \l_tmpa_tl
2332 \IfPackageLoadedTF { varwidth }
2333 { \@@_patch_preamble_iv_iv:n { #2 } { varwidth } }
2334 {
2335   \@@_error_or_warning:n { varwidth~not~loaded }
2336   \@@_patch_preamble_iv_iv:n { #2 } { minipage }
2337 }
2338 }

```

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.

```

2339 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2340 {
2341   \str_if_eq:nnTF { #3 } { s }
2342   { \@@_patch_preamble_vi_i:nnnn { #1 } { #4 } }
2343   { \@@_patch_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2344 }

```

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.

```

2345 \cs_new_protected:Npn \@@_patch_preamble_vi_i:nnnn #1 #2
2346 {
2347   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2348   \tl_gclear:N \g_@@_pre_cell_tl
2349   \tl_gput_right:Nn \g_@@_preamble_tl
2350   {
2351     > {
2352       \dim_set:Nn \l_@@_col_width_dim { #2 }
2353       \@@_cell_begin:w
2354       \str_set:Nn \l_@@_hpos_cell_str { c }
2355     }
2356     c
2357     < {
2358       \@@_cell_end_for_w_s:
2359       #1
2360       \@@_adjust_size_box:
2361       \box_use_drop:N \l_@@_cell_box
2362     }
2363   }
2364   \int_gincr:N \c@jCol
2365   \@@_patch_preamble_xi:n
2366 }

```

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

```

2367 \cs_new_protected:Npn \@@_patch_preamble_vi_ii:nnnn #1 #2 #3 #4
2368 {
2369   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2370   \tl_gclear:N \g_@@_pre_cell_tl
2371   \tl_gput_right:Nn \g_@@_preamble_tl
2372   {
2373     > {

```

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.

```

2374         \dim_set:Nn \l_@@_col_width_dim { #4 }
2375         \hbox_set:Nw \l_@@_cell_box
2376         \@@_cell_begin:w
2377         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2378     }
2379     c
2380     < {
2381         \@@_cell_end:
2382         \hbox_set_end:
2383         % The following line is probably pointless
2384         % \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2385         #1
2386         \@@_adjust_size_box:
2387         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2388     }
2389 }

```

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

```

2390     \int_gincr:N \c@jCol
2391     \@@_patch_preamble_xi:n
2392 }

2393 \cs_new_protected:Npn \@@_special_W:
2394 {
2395     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2396     { \@@_warning:n { W-warning } }
2397 }

```

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).

```

2398 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2399 {
2400     \str_if_eq:nnTF { #1 } { [ ]
2401         { \@@_patch_preamble_vii_i:w [ ]
2402             { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2403         }
2404     \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2405     { \@@_patch_preamble_vii_ii:n { #1 } }
2406     \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2407     {
2408         \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2409         {
2410             \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2411             \tl_gclear:N \g_@@_pre_cell_tl
2412             \tl_gput_right:Nn \g_@@_preamble_tl
2413             {
2414                 > {
2415                     \@@_cell_begin:w
2416                     \keys_set:nn { siunitx } { #1 }
2417                     \siunitx_cell_begin:w
2418                 }
2419                 c
2420                 < { \siunitx_cell_end: \@@_cell_end: }
2421             }

```

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

```

2422     \int_gincr:N \c@jCol
2423     \@@_patch_preamble_xi:n
2424 }
2425 { \@@_fatal:n { Version-of-siunitx-too-old } }
2426 }

```


For (, [and \{.

```

2427 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2428 {
2429   \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.

```

2430   \int_compare:nNnTF \c@jCol = \c_zero_int
2431   {
2432     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2433     {

```

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

```

2434       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2435       \tl_gset:Nn \g_@@_right_delim_tl { . }
2436       \@@_patch_preamble:n #2
2437     }
2438     {
2439       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2440       \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2441     }
2442   }
2443   { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2444 }
2445 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2446 {
2447   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2448   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2449   \tl_if_in:nnTF { ( [ \{ ) ] \} } \left \right } { #2 }
2450   {
2451     \@@_error:nn { delimiter~after~opening } { #2 }
2452     \@@_patch_preamble:n
2453   }
2454   { \@@_patch_preamble:n #2 }
2455 }

```

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

```

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

```

```

2479     }
2480   }
2481 }
2482 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2483 {
2484   \tl_if_eq:nnTF { \q_stop } { #3 }
2485   {
2486     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2487     {
2488       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2489       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2490       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2491       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2492     }
2493     {
2494       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2495       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2496       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2497       \@@_error:nn { double~closing~delimiter } { #2 }
2498     }
2499   }
2500   {
2501     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2502     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2503     \@@_error:nn { double~closing~delimiter } { #2 }
2504     \@@_patch_preamble:n #3
2505   }
2506 }

```

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.

```

2507 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2508 {
2509   \str_if_eq:nnTF { #1 } { [ ]
2510     { \@@_patch_preamble_x_i:w [ ]
2511       { \@@_patch_preamble_x_i:w [ ] #1 }
2512     }
2513   \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2514     { \@@_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).

```

2515 \keys_define:nn { WithArrows / X-column }
2516 { 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.

```

2517 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2518 {

```

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).

```

2519   \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).

```

2520   \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`.

```

2521 \int_zero_new:N \l_@@_weight_int
2522 \int_set:Nn \l_@@_weight_int { 1 }
2523 \tl_set:Nn \l_tmpa_tl { #1 }
2524 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2525 \@@_keys_p_column:V \l_tmpa_tl
2526 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2527 \int_compare:nNnT \l_@@_weight_int < 0
2528 {
2529   \@@_error_or_warning:n { negative~weight }
2530   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2531 }
2532 \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).

```

2533 \bool_if:NTF \l_@@_X_columns_aux_bool
2534 {
2535   \exp_args:Nnx
2536   \@@_patch_preamble_iv_iv:nn
2537   { \l_@@_weight_int \l_@@_X_columns_dim }
2538   { minipage }
2539 }
2540 {
2541   \tl_gput_right:Nn \g_@@_preamble_tl
2542   {
2543     > {
2544       \@@_cell_begin:w
2545       \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`.

```

2546 \NotEmpty

```

The following code will nullify the box of the cell.

```

2547 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2548 { \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`.

```

2549 \begin { minipage } { 5 cm } \arraybackslash
2550 }
2551 c
2552 < {
2553   \end { minipage }
2554   \@@_cell_end:
2555 }
2556 }
2557 \int_gincr:N \c@jCol
2558 \@@_patch_preamble_xi:n
2559 }
2560 }

```

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 `@{...}`.

```

2561 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2562 {

```

```

2563 \str_if_eq:nnTF { #1 } { < }
2564 \@@_patch_preamble_xiii:n
2565 {
2566   \str_if_eq:nnTF { #1 } { @ }
2567   \@@_patch_preamble_xv:n
2568   {
2569     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2570     {
2571       \tl_gput_right:Nn \g_@@_preamble_tl
2572       { ! { \skip_horizontal:N \arrayrulewidth } }
2573     }
2574     {
2575       \exp_args:NNx
2576       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2577       {
2578         \tl_gput_right:Nn \g_@@_preamble_tl
2579         { ! { \skip_horizontal:N \arrayrulewidth } }
2580       }
2581     }
2582     \@@_patch_preamble:n { #1 }
2583   }
2584 }
2585 }
2586 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2587 {
2588   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2589   \@@_patch_preamble_xi:n
2590 }

```

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.

```

2591 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2592 {
2593   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2594   {
2595     \tl_gput_right:Nn \g_@@_preamble_tl
2596     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2597   }
2598   {
2599     \exp_args:NNx
2600     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2601     {
2602       \tl_gput_right:Nn \g_@@_preamble_tl
2603       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2604     }
2605     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2606   }
2607   \@@_patch_preamble:n
2608 }
2609 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2610 {
2611   \group_begin:
2612   \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2613   \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2614   \@temptokena { #2 }
2615   \@tempswatrue
2616   \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2617   \tl_gclear:N \g_@@_preamble_tl
2618   \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
2619   \group_end:
2620   \tl_set_eq:NN #1 \g_@@_preamble_tl
2621 }

```

12 The redefinition of `\multicolumn`

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

```
2622 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2623 {
```

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`.

```
2624 \multispan { #1 }
2625 \begingroup
2626 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
2627 \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2628 \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
```

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

```
2629 \@temptokena = { #2 }
2630 \@tempswatrue
2631 \@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`.

```
2632 \tl_gclear:N \g_@@_preamble_tl
2633 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
```

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

```
2634 \exp_args:NV \mkpream \g_@@_preamble_tl
2635 \addtopreamble \empty
2636 \endgroup
```

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

```
2637 \int_compare:nNnT { #1 } > 1
2638 {
2639   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2640   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2641   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2642   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2643   {
2644     {
2645       \int_compare:nNnTF \c@jCol = 0
2646       { \int_eval:n { \c@iRow + 1 } }
2647       { \int_use:N \c@iRow }
2648     }
2649     { \int_eval:n { \c@jCol + 1 } }
2650     {
2651       \int_compare:nNnTF \c@jCol = 0
2652       { \int_eval:n { \c@iRow + 1 } }
2653       { \int_use:N \c@iRow }
2654     }
2655     { \int_eval:n { \c@jCol + #1 } }
2656     { } % for the name of the block
2657   }
2658 }
```

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

```
2659 \cs_set:Npn \@sharp { #3 }
2660 \@arstrut
2661 \@preamble
2662 \null
```

We add some lines.

```

2663 \int_gadd:Nn \c@jCol { #1 - 1 }
2664 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2665   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2666 \ignorespaces
2667 }

```

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`.

```

2668 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2669 {
2670   \str_case:nnF { #1 }
2671   {
2672     c { \@@_patch_m_preamble_i:n #1 }
2673     l { \@@_patch_m_preamble_i:n #1 }
2674     r { \@@_patch_m_preamble_i:n #1 }
2675     > { \@@_patch_m_preamble_ii:nn #1 }
2676     ! { \@@_patch_m_preamble_ii:nn #1 }
2677     @ { \@@_patch_m_preamble_ii:nn #1 }
2678     | { \@@_patch_m_preamble_iii:n #1 }
2679     p { \@@_patch_m_preamble_iv:nnn t #1 }
2680     m { \@@_patch_m_preamble_iv:nnn c #1 }
2681     b { \@@_patch_m_preamble_iv:nnn b #1 }
2682     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2683     \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2684     \q_stop { }
2685   }
2686   {
2687     \tl_if_eq:nnT { #1 } { S }
2688     { \@@_fatal:n { unknown~column~type~S } }
2689     { \@@_fatal:nn { unknown~column~type } { #1 } }
2690   }
2691 }

```

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

```

2692 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2693 {
2694   \tl_gput_right:Nn \g_@@_preamble_tl
2695   {
2696     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2697     #1
2698     < \@@_cell_end:
2699   }

```

We test for the presence of a `<`.

```

2700 \@@_patch_m_preamble_x:n
2701 }

```

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

```

2702 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2703 {
2704   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2705   \@@_patch_m_preamble:n
2706 }

```

For `|`

```

2707 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2708 {
2709   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2710   \@@_patch_m_preamble:n
2711 }

```

For p, m and b

```

2712 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2713 {
2714   \tl_gput_right:Nn \g_@@_preamble_tl
2715   {
2716     > {
2717       \@@_cell_begin:w
2718       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2719       \mode_leave_vertical:
2720       \arraybackslash
2721       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2722     }
2723     c
2724     < {
2725       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2726       \end { minipage }
2727       \@@_cell_end:
2728     }
2729   }

```

We test for the presence of a <.

```

2730   \@@_patch_m_preamble_x:n
2731 }

```

For w and W

```

2732 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2733 {
2734   \tl_gput_right:Nn \g_@@_preamble_tl
2735   {
2736     > {
2737       \dim_set:Nn \l_@@_col_width_dim { #4 }
2738       \hbox_set:Nw \l_@@_cell_box
2739       \@@_cell_begin:w
2740       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2741     }
2742     c
2743     < {
2744       \@@_cell_end:
2745       \hbox_set_end:
2746       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2747       #1
2748       \@@_adjust_size_box:
2749       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2750     }
2751   }

```

We test for the presence of a <.

```

2752   \@@_patch_m_preamble_x:n
2753 }

```

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

```

2754 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2755 {
2756   \str_if_eq:nnTF { #1 } { < }
2757   \@@_patch_m_preamble_ix:n
2758   { \@@_patch_m_preamble:n { #1 } }
2759 }
2760 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2761 {
2762   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2763   \@@_patch_m_preamble_x:n
2764 }

```

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`).

```

2765 \cs_new_protected:Npn \@@_put_box_in_flow:
2766 {
2767   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2768   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2769   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2770     { \box_use_drop:N \l_tmpa_box }
2771     \@@_put_box_in_flow_i:
2772 }

```

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).

```

2773 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2774 {
2775   \pgfpicture
2776     \@@_qpoint:n { row - 1 }
2777     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2778     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2779     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2780     \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).

```

2781   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2782   {
2783     \int_set:Nn \l_tmpa_int
2784     {
2785       \str_range:Nnn
2786         \l_@@_baseline_tl
2787         6
2788         { \tl_count:V \l_@@_baseline_tl }
2789     }
2790     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2791   }
2792   {
2793     \str_case:VnF \l_@@_baseline_tl
2794     {
2795       { t } { \int_set:Nn \l_tmpa_int 1 }
2796       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2797     }
2798     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2799     \bool_lazy_or:nnT
2800     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2801     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2802     {
2803       \@@_error:n { bad~value~for~baseline }
2804       \int_set:Nn \l_tmpa_int 1
2805     }
2806     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2807     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2808   }
2809   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2810   \endpgfpicture
2811   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2812   \box_use_drop:N \l_tmpa_box
2813 }

```


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).

```
2814 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2815 {
```

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.

```
2816 \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2817 {
2818   \box_set_wd:Nn \l_@@_the_array_box
2819   { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2820 }
```

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

```
2821 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2822 \bool_if:NT \l_@@_caption_above_bool
2823 {
2824   \tl_if_empty:NF \l_@@_caption_tl
2825   {
2826     \bool_set_false:N \g_@@_caption_finished_bool
2827     \int_gzero:N \c@tabularnote
2828     \@@_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).

```
2829     \int_compare:nNnT \g_@@_notes_caption_int > 0
2830     {
2831       \tl_gput_right:Nx \g_@@_aux_tl
2832       {
2833         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2834         { \int_use:N \g_@@_notes_caption_int }
2835       }
2836       \int_gzero:N \g_@@_notes_caption_int
2837     }
2838   }
2839 }
```

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

```
2840 \hbox
2841 {
2842   \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.

```
2843 \@@_create_extra_nodes:
2844 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2845 }
```

We don't do the following test with `\c@tabularnote` because the value of that counter is not reliable when the command `\ttabbox` of `floatrow` is used (because `\ttabbox` de-activate `\stepcounter` because it compiles several times its tabular).

```
2846 \bool_lazy_any:nT
2847 {
2848   { ! \seq_if_empty_p:N \g_@@_notes_seq }
2849   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2850   { ! \tl_if_empty_p:N \g_@@_tabularnote_tl }
2851 }
```

```

2852 \@@_insert_tabularnotes:
2853 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2854 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2855 \end { minipage }
2856 }

```

```

2857 \cs_new_protected:Npn \@@_insert_caption:
2858 {
2859 \tl_if_empty:NF \l_@@_caption_tl
2860 {
2861 \cs_if_exist:NTF \@capttype
2862 { \@@_insert_caption_i: }
2863 { \@@_error:n { caption-outside~float } }
2864 }
2865 }

```

```

2866 \cs_new_protected:Npn \@@_insert_caption_i:
2867 {
2868 \group_begin:

```

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

```

2869 \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.

```

2870 \IfPackageLoadedTF { floatrow }
2871 { \cs_set_eq:NN \@makecaption \FR@makecaption }
2872 { }
2873 \tl_if_empty:NTF \l_@@_short_caption_tl
2874 { \caption }
2875 { \caption [ \l_@@_short_caption_tl ] }
2876 { \l_@@_caption_tl }
2877 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2878 \group_end:
2879 }

```

```

2880 \cs_new_protected:Npn \@@_tabularnote_error:n #1
2881 {
2882 \@@_error_or_warning:n { tabularnote~below~the~tabular }
2883 \@@_gredirect_none:n { tabularnote~below~the~tabular }
2884 }

```

```

2885 \cs_new_protected:Npn \@@_insert_tabularnotes:
2886 {
2887 \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2888 \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2889 \skip_vertical:N 0.65ex

```

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

```

2890 \group_begin:
2891 \l_@@_notes_code_before_tl
2892 \tl_if_empty:NF \g_@@_tabularnote_tl
2893 {
2894 \g_@@_tabularnote_tl \par
2895 \tl_gclear:N \g_@@_tabularnote_tl
2896 }

```

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.

```

2897 \int_compare:nNnT \c@tabularnote > 0
2898 {
2899 \bool_if:NTF \l_@@_notes_para_bool

```

```

2900     {
2901         \begin { tabularnotes* }
2902         \seq_map_inline:Nn \g_@@_notes_seq
2903         { \@@_one_tabularnote:nn ##1 }
2904         \strut
2905         \end { tabularnotes* }

```

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

```

2906     \par
2907 }
2908 {
2909     \tabularnotes
2910     \seq_map_inline:Nn \g_@@_notes_seq
2911     { \@@_one_tabularnote:nn ##1 }
2912     \strut
2913     \endtabularnotes
2914 }
2915 }
2916 \unskip
2917 \group_end:
2918 \bool_if:NT \l_@@_notes_bottomrule_bool
2919 {
2920     \IfPackageLoadedTF { booktabs }
2921     {

```

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

```

2922         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2923         { \CT@arc@ \hrule height \heavyrulewidth }
2924     }
2925     { \@@_error_or_warning:n { bottomrule-without~booktabs } }
2926 }
2927 \l_@@_notes_code_after_tl
2928 \seq_gclear:N \g_@@_notes_seq
2929 \seq_gclear:N \g_@@_notes_in_caption_seq
2930 \int_gzero:N \c@tabularnote
2931 }

```

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

```

2932 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
2933 {
2934     \tl_if_novalue:nTF { #1 }
2935     { \item }
2936     { \item [ \@@_notes_label_in_list:n { #1 } ] }
2937 }

```

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`.

```

2938 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2939 {
2940     \pgfpicture
2941     \@@_qpoint:n { row - 1 }
2942     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2943     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2944     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2945     \endpgfpicture
2946     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2947     \int_compare:nNnT \l_@@_first_row_int = 0

```

```

2948 {
2949   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2950   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2951 }
2952 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2953 }

```

Now, the general case.

```

2954 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2955 {

```

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

```

2956   \tl_if_eq:NnT \l_@@_baseline_tl { t }
2957   { \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`.

```

2958   \pgfpicture
2959   \@@_qpoint:n { row - 1 }
2960   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2961   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2962   {
2963     \int_set:Nn \l_tmpa_int
2964     {
2965       \str_range:Nnn
2966       \l_@@_baseline_tl
2967       6
2968       { \tl_count:V \l_@@_baseline_tl }
2969     }
2970     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2971   }
2972   {
2973     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2974     \bool_lazy_or:nnT
2975     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2976     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2977     {
2978       \@@_error:n { bad~value~for~baseline }
2979       \int_set:Nn \l_tmpa_int 1
2980     }
2981     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2982   }
2983   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2984   \endpgfpicture
2985   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2986   \int_compare:nNnT \l_@@_first_row_int = 0
2987   {
2988     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2989     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2990   }
2991   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2992 }

```

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.

```

2993 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2994 {

```

We will compute the real width of both delimiters used.

```

2995   \dim_zero_new:N \l_@@_real_left_delim_dim
2996   \dim_zero_new:N \l_@@_real_right_delim_dim
2997   \hbox_set:Nn \l_tmpb_box
2998   {

```

```

2999     \c_math_toggle_token
3000     \left #1
3001     \vcenter
3002     {
3003         \vbox_to_ht:nn
3004         { \box_ht_plus_dp:N \l_tmpa_box }
3005         { }
3006     }
3007     \right .
3008     \c_math_toggle_token
3009 }
3010 \dim_set:Nn \l_@@_real_left_delim_dim
3011 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3012 \hbox_set:Nn \l_tmpb_box
3013 {
3014     \c_math_toggle_token
3015     \left .
3016     \vbox_to_ht:nn
3017     { \box_ht_plus_dp:N \l_tmpa_box }
3018     { }
3019     \right #2
3020     \c_math_toggle_token
3021 }
3022 \dim_set:Nn \l_@@_real_right_delim_dim
3023 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3024     \skip_horizontal:N \l_@@_left_delim_dim
3025     \skip_horizontal:N -\l_@@_real_left_delim_dim
3026     \@@_put_box_in_flow:
3027     \skip_horizontal:N \l_@@_right_delim_dim
3028     \skip_horizontal:N -\l_@@_real_right_delim_dim
3029 }

```

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).

```

3030 \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).

```

3031 {
3032     \peek_remove_spaces:n
3033     {
3034         \peek_meaning:NTF \end
3035         \@@_analyze_end:Nn
3036         {
3037             \@@_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`).

```

3038         \@@_array:V \g_@@_preamble_tl
3039     }
3040 }
3041 }
3042 {
3043     \@@_create_col_nodes:
3044     \endarray
3045 }

```

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

```
3046 \NewDocumentEnvironment { @@-light-syntax } { b }
3047 {
```

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`.

```
3048 \tl_if_empty:nT { #1 } { @@_fatal:n { empty-environment } }
3049 \tl_map_inline:nn { #1 }
3050 {
3051   \str_if_eq:nnT { ##1 } { & }
3052   { @@_fatal:n { ampersand-in~light-syntax } }
3053   \str_if_eq:nnT { ##1 } { \ }
3054   { @@_fatal:n { double-backslash-in~light-syntax } }
3055 }
```

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`.

```
3056 @@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3057 }
```

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.

```
3058 {
3059   @@_create_col_nodes:
3060   \endarray
3061 }

3062 \cs_new_protected:Npn @@_light_syntax_i:w #1\CodeAfter #2\q_stop
3063 {
3064   \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).

```
3065 \seq_clear_new:N \l_@@_rows_seq
```

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

```
3066 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3067 \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3068 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3069 \tl_if_empty:NF \l_tmpa_tl
3070 { \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.

```
3071 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3072 { \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`).

```
3073 \tl_clear_new:N \l_@@_new_body_tl
3074 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3075 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3076 \@@_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).

```
3077 \seq_map_inline:Nn \l_@@_rows_seq
3078 {
3079   \tl_put_right:Nn \l_@@_new_body_tl { \\ }
3080   \@@_line_with_light_syntax:n { ##1 }
3081 }
3082 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3083 {
3084   \int_set:Nn \l_@@_last_col_int
3085     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3086 }
```

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.

```
3087 \@@_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`).

```
3088 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3089 }
3090 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3091 {
3092   \seq_clear_new:N \l_@@_cells_seq
3093   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3094   \int_set:Nn \l_@@_nb_cols_int
3095     {
3096       \int_max:nn
3097         \l_@@_nb_cols_int
3098         { \seq_count:N \l_@@_cells_seq }
3099     }
3100   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3101   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3102   \seq_map_inline:Nn \l_@@_cells_seq
3103     { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3104 }
3105 \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`.

```
3106 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3107 {
3108   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3109   { \@@_fatal:n { empty-environment } }
3110 }
```

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

```
3110 \end { #2 }
3111 }
```

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).

```
3112 \cs_new:Npn \@@_create_col_nodes:
3113 {
3114   \crrc
3115   \int_compare:nNnT \l_@@_first_col_int = 0
3116   {
```

```

3117 \omit
3118 \hbox_overlap_left:n
3119 {
3120   \bool_if:NT \l_@@_code_before_bool
3121     { \pgfsys@markposition { \@@_env: - col - 0 } }
3122   \pgfpicture
3123   \pgfrememberpicturepositiononpagetrue
3124   \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3125   \str_if_empty:NF \l_@@_name_str
3126     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3127   \endpgfpicture
3128   \skip_horizontal:N 2\col@sep
3129   \skip_horizontal:N \g_@@_width_first_col_dim
3130 }
3131 &
3132 }
3133 \omit

```

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

```

3134 \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`).

```

3135 \int_compare:nNnTF \l_@@_first_col_int = 0
3136 {
3137   \bool_if:NT \l_@@_code_before_bool
3138   {
3139     \hbox
3140     {
3141       \skip_horizontal:N -0.5\arrayrulewidth
3142       \pgfsys@markposition { \@@_env: - col - 1 }
3143       \skip_horizontal:N 0.5\arrayrulewidth
3144     }
3145   }
3146   \pgfpicture
3147   \pgfrememberpicturepositiononpagetrue
3148   \pgfcoordinate { \@@_env: - col - 1 }
3149     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3150   \str_if_empty:NF \l_@@_name_str
3151     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3152   \endpgfpicture
3153 }
3154 {
3155   \bool_if:NT \l_@@_code_before_bool
3156   {
3157     \hbox
3158     {
3159       \skip_horizontal:N 0.5\arrayrulewidth
3160       \pgfsys@markposition { \@@_env: - col - 1 }
3161       \skip_horizontal:N -0.5\arrayrulewidth
3162     }
3163   }
3164   \pgfpicture
3165   \pgfrememberpicturepositiononpagetrue
3166   \pgfcoordinate { \@@_env: - col - 1 }
3167     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3168   \str_if_empty:NF \l_@@_name_str
3169     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3170   \endpgfpicture
3171 }

```

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.

```

3172 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3173 \bool_if:NF \l_@@_auto_columns_width_bool
3174 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3175 {
3176   \bool_lazy_and:nnTF
3177     \l_@@_auto_columns_width_bool
3178     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3179     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3180     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3181     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3182 }
3183 \skip_horizontal:N \g_tmpa_skip
3184 \hbox
3185 {
3186   \bool_if:NT \l_@@_code_before_bool
3187   {
3188     \hbox
3189     {
3190       \skip_horizontal:N -0.5\arrayrulewidth
3191       \pgfsys@markposition { \@@_env: - col - 2 }
3192       \skip_horizontal:N 0.5\arrayrulewidth
3193     }
3194   }
3195   \pgfpicture
3196   \pgfrememberpicturerepositiononpagetrue
3197   \pgfcoordinate { \@@_env: - col - 2 }
3198   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3199   \str_if_empty:NF \l_@@_name_str
3200   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3201   \endpgfpicture
3202 }

```

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.

```

3203 \int_gset:Nn \g_tmpa_int 1
3204 \bool_if:NTF \g_@@_last_col_found_bool
3205 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3206 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3207 {
3208   &
3209   \omit
3210   \int_gincr:N \g_tmpa_int

```

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

```

3211 \skip_horizontal:N \g_tmpa_skip
3212 \bool_if:NT \l_@@_code_before_bool
3213 {
3214   \hbox
3215   {
3216     \skip_horizontal:N -0.5\arrayrulewidth
3217     \pgfsys@markposition
3218     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3219     \skip_horizontal:N 0.5\arrayrulewidth
3220   }
3221 }

```

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

```

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

```

```

3227     {
3228         \pgfnodealias
3229         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3230         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3231     }
3232 \endpgfpicture
3233 }

```

```

3234 &
3235 \omit

```

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

```

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

```

```

3251     \hbox
3252     {
3253         \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.

```

3254         \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3255         { \skip_horizontal:N -\arraycolsep }
3256     \pgfsys@markposition
3257     { \@@_env: - col - \int_eval:n {
3258         \g_tmpa_int + 1 } }
3259     \skip_horizontal:N 0.5\arrayrulewidth
3260     \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3261     { \skip_horizontal:N \arraycolsep }
3262 }
3263 }
3264 \pgfpicture
3265 \pgfrememberpicturepositiononpagetrue
3266 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3267 {
3268     \bool_lazy_and:nnTF \l_@@_Matrix_bool \g_@@_NiceArray_bool
3269     {
3270         \pgfpoint
3271         { - 0.5 \arrayrulewidth - \arraycolsep }
3272         \c_zero_dim
3273     }
3274     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3275 }
3276 \str_if_empty:NF \l_@@_name_str
3277 {
3278     \pgfnodealias
3279     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3280     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3281 }
3282 \endpgfpicture

```

```

3283 \bool_if:NT \g_@@_last_col_found_bool
3284 {
3285   \hbox_overlap_right:n
3286   {
3287     \skip_horizontal:N \g_@@_width_last_col_dim
3288     \bool_if:NT \l_@@_code_before_bool
3289     {
3290       \pgfsys@markposition
3291       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3292     }
3293     \pgfpicture
3294     \pgfrememberpicturepositiononpagetrue
3295     \pgfcoordinate
3296     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3297     \pgfpointorigin
3298     \str_if_empty:NF \l_@@_name_str
3299     {
3300       \pgfnodealias
3301       {
3302         \l_@@_name_str - col
3303         - \int_eval:n { \g_@@_col_total_int + 1 }
3304       }
3305       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3306     }
3307     \endpgfpicture
3308   }
3309 }
3310 \cr
3311 }

```

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

```

3312 \tl_const:Nn \c_@@_preamble_first_col_tl
3313 {
3314   >
3315   {

```

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).

```

3316 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3317 \bool_gset_true:N \g_@@_after_col_zero_bool
3318 \@@_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.

```

3319 \hbox_set:Nw \l_@@_cell_box
3320 \@@_math_toggle_token:
3321 \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”.

```

3322 \bool_lazy_and:nnT
3323 { \int_compare_p:nNn \c@iRow > 0 }
3324 {
3325   \bool_lazy_or_p:nn
3326   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3327   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3328 }
3329 {
3330   \l_@@_code_for_first_col_tl
3331   \xglobal \colorlet { nicematrix-first-col } { . }
3332 }
3333 }

```

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`.

```

3334     l
3335     <
3336     {
3337         \@@_math_toggle_token:
3338         \hbox_set_end:
3339         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3340         \@@_adjust_size_box:
3341         \@@_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.

```

3342         \dim_gset:Nn \g_@@_width_first_col_dim
3343         { \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.

```

3344         \hbox_overlap_left:n
3345         {
3346             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3347             \@@_node_for_cell:
3348             { \box_use_drop:N \l_@@_cell_box }
3349             \skip_horizontal:N \l_@@_left_delim_dim
3350             \skip_horizontal:N \l_@@_left_margin_dim
3351             \skip_horizontal:N \l_@@_extra_left_margin_dim
3352         }
3353         \bool_gset_false:N \g_@@_empty_cell_bool
3354         \skip_horizontal:N -2\col@sep
3355     }
3356 }

```

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

```

3357 \tl_const:Nn \c_@@_preamble_last_col_tl
3358 {
3359     >
3360     {

```

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).

```

3361         \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.

```

3362         \bool_gset_true:N \g_@@_last_col_found_bool
3363         \int_gincr:N \c@jCol
3364         \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.

```

3365         \hbox_set:Nw \l_@@_cell_box
3366         \@@_math_toggle_token:
3367         \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”.

```

3368         \int_compare:nNnT \c@iRow > 0
3369         {
3370             \bool_lazy_or:nnT
3371             { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3372             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3373             {
3374                 \l_@@_code_for_last_col_tl
3375                 \xglobal \colorlet { nicematrix-last-col } { . }
3376             }
3377         }
3378     }
3379     l

```

```

3380 <
3381 {
3382   \@@_math_toggle_token:
3383   \hbox_set_end:
3384   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3385   \@@_adjust_size_box:
3386   \@@_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.

```

3387   \dim_gset:Nn \g_@@_width_last_col_dim
3388   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3389   \skip_horizontal:N -2\col@sep

```

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

```

3390   \hbox_overlap_right:n
3391   {
3392     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3393     {
3394       \skip_horizontal:N \l_@@_right_delim_dim
3395       \skip_horizontal:N \l_@@_right_margin_dim
3396       \skip_horizontal:N \l_@@_extra_right_margin_dim
3397       \@@_node_for_cell:
3398     }
3399   }
3400   \bool_gset_false:N \g_@@_empty_cell_bool
3401 }
3402 }

```

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.

```

3403 \NewDocumentEnvironment { NiceArray } { }
3404 {
3405   \bool_gset_true:N \g_@@_NiceArray_bool
3406   \str_if_empty:NT \g_@@_name_env_str
3407   { \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).

```

3408   \NiceArrayWithDelims . .
3409 }
3410 { \endNiceArrayWithDelims }

```

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

```

3411 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3412 {
3413   \NewDocumentEnvironment { #1 NiceArray } { }
3414   {
3415     \bool_gset_false:N \g_@@_NiceArray_bool
3416     \str_if_empty:NT \g_@@_name_env_str
3417     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3418     \@@_test_if_math_mode:
3419     \NiceArrayWithDelims #2 #3
3420   }
3421   { \endNiceArrayWithDelims }
3422 }
3423 \@@_def_env:nnn p ( )
3424 \@@_def_env:nnn b [ ]
3425 \@@_def_env:nnn B \{ \}
3426 \@@_def_env:nnn v | |
3427 \@@_def_env:nnn V \| \|

```

13 The environment {NiceMatrix} and its variants

```

3428 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3429 {
3430   \bool_set_true:N \l_@@_Matrix_bool
3431   \use:c { #1 NiceArray }
3432   {
3433     *
3434     {
3435       \int_case:nnF \l_@@_last_col_int
3436       {
3437         { -2 } { \c@MaxMatrixCols }
3438         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3439       }
3440       { \int_eval:n { \l_@@_last_col_int - 1 } }
3441     }
3442     { #2 }
3443   }
3444 }
3445 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3446 \clist_map_inline:nn { p , b , B , v , V }
3447 {
3448   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3449   {
3450     \bool_gset_false:N \g_@@_NiceArray_bool
3451     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3452     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3453     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3454   }
3455   { \use:c { end #1 NiceArray } }
3456 }

```

We define also an environment {NiceMatrix}

```

3457 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3458 {
3459   \bool_gset_false:N \g_@@_NiceArray_bool
3460   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3461   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3462   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3463 }
3464 { \endNiceArray }

```

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

```

3465 \cs_new_protected:Npn \@@_NotEmpty:
3466 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3467 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3468 {

```

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.

```

3469   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3470   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3471   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3472   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3473   \int_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim

```

```

3474 {
3475   \bool_if:NT \l_@@_hvlines_bool
3476   {
3477     \bool_set_true:N \l_@@_except_borders_bool
3478     % we should try to be more efficient in the number of lines of code here
3479     \tl_if_empty:NTF \l_@@_rules_color_tl
3480     {
3481       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3482       {
3483         \@@_stroke_block:nnn
3484         { rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim }
3485         { 1-1 }
3486         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3487       }
3488     }
3489     {
3490       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3491       {
3492         \@@_stroke_block:nnn
3493         {
3494           rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3495           draw = \l_@@_rules_color_tl
3496         }
3497         { 1-1 }
3498         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3499       }
3500     }
3501   }
3502 }
3503 \tl_if_empty:NF \l_@@_short_caption_tl
3504 {
3505   \tl_if_empty:NT \l_@@_caption_tl
3506   {
3507     \@@_error_or_warning:n { short-caption~without~caption }
3508     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3509   }
3510 }
3511 \tl_if_empty:NF \l_@@_label_tl
3512 {
3513   \tl_if_empty:NT \l_@@_caption_tl
3514   { \@@_error_or_warning:n { label~without~caption } }
3515 }
3516 \NewDocumentEnvironment { TabularNote } { b }
3517 {
3518   \bool_if:NTF \l_@@_in_code_after_bool
3519   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3520   {
3521     \tl_if_empty:NF \g_@@_tabularnote_tl
3522     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3523     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3524   }
3525 }
3526 { }
3527 \bool_set_true:N \l_@@_NiceTabular_bool
3528 \NiceArray { #2 }
3529 }
3530 { \endNiceArray }
3531 \cs_set_protected:Npn \@@_newcolumnntype #1
3532 {
3533   \cs_if_free:cT { NC @ find @ #1 }
3534   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3535   \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }

```

```

3536 \peek_meaning:N\TF [
3537   { \newcol@ #1 }
3538   { \newcol@ #1 [ 0 ] }
3539 ]

```

```

3540 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3541 {

```

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}`).

```

3542 \IfPackageLoadedTF { tabularx }
3543   { \newcolumnntype { X } { \@@_X } }
3544   { }
3545 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3546 \dim_zero_new:N \l_@@_width_dim
3547 \dim_set:Nn \l_@@_width_dim { #1 }
3548 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3549 \bool_set_true:N \l_@@_NiceTabular_bool
3550 \NiceArray { #3 }
3551 }
3552 { \endNiceArray }

```

```

3553 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3554 {
3555   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3556   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3557   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3558   \bool_set_true:N \l_@@_NiceTabular_bool
3559   \NiceArray { #3 }
3560 }
3561 { \endNiceArray }

```

15 After the construction of the array

```

3562 \cs_new_protected:Npn \@@_after_array:
3563 {
3564   \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.

```

3565 \bool_if:NT \g_@@_last_col_found_bool
3566   { \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`.

```

3567 \bool_if:NT \l_@@_last_col_without_value_bool
3568   { \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.

```

3569 \bool_if:NT \l_@@_last_row_without_value_bool
3570   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3571 \tl_gput_right:Nx \g_@@_aux_tl
3572 {
3573   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3574   {
3575     \int_use:N \l_@@_first_row_int ,
3576     \int_use:N \c@iRow ,

```



```

3577         \int_use:N \g_@@_row_total_int ,
3578         \int_use:N \l_@@_first_col_int ,
3579         \int_use:N \c@jCol ,
3580         \int_use:N \g_@@_col_total_int
3581     }
3582 }

```

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`).

```

3583 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3584 {
3585     \tl_gput_right:Nx \g_@@_aux_tl
3586     {
3587         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3588         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3589     }
3590 }
3591 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3592 {
3593     \tl_gput_right:Nx \g_@@_aux_tl
3594     {
3595         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3596         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3597         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3598         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3599     }
3600 }

```

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

```

3601 \@@_create_diag_nodes:

```

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

```

3602 \pgfpicture
3603 \int_step_inline:nn \c@iRow
3604 {
3605     \pgfnodealias
3606     { \@@_env: - ##1 - last }
3607     { \@@_env: - ##1 - \int_use:N \c@jCol }
3608 }
3609 \int_step_inline:nn \c@jCol
3610 {
3611     \pgfnodealias
3612     { \@@_env: - last - ##1 }
3613     { \@@_env: - \int_use:N \c@iRow - ##1 }
3614 }
3615 \str_if_empty:NF \l_@@_name_str
3616 {
3617     \int_step_inline:nn \c@iRow
3618     {
3619         \pgfnodealias
3620         { \l_@@_name_str - ##1 - last }
3621         { \@@_env: - ##1 - \int_use:N \c@jCol }
3622     }
3623     \int_step_inline:nn \c@jCol
3624     {
3625         \pgfnodealias
3626         { \l_@@_name_str - last - ##1 }
3627         { \@@_env: - \int_use:N \c@iRow - ##1 }
3628     }
3629 }
3630 \endpgfpicture

```

By default, the diagonal lines will be parallelized¹¹. 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.

```

3631 \bool_if:NT \l_@@_parallelize_diags_bool
3632 {
3633     \int_gzero_new:N \g_@@_ddots_int
3634     \int_gzero_new:N \g_@@_iddots_int

```

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

```

3635     \dim_gzero_new:N \g_@@_delta_x_one_dim
3636     \dim_gzero_new:N \g_@@_delta_y_one_dim
3637     \dim_gzero_new:N \g_@@_delta_x_two_dim
3638     \dim_gzero_new:N \g_@@_delta_y_two_dim
3639 }
3640 \int_zero_new:N \l_@@_initial_i_int
3641 \int_zero_new:N \l_@@_initial_j_int
3642 \int_zero_new:N \l_@@_final_i_int
3643 \int_zero_new:N \l_@@_final_j_int
3644 \bool_set_false:N \l_@@_initial_open_bool
3645 \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.

```

3646 \bool_if:NT \l_@@_small_bool
3647 {
3648     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3649     \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.

```

3650     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3651     { 0.6 \l_@@_xdots_shorten_start_dim }
3652     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3653     { 0.6 \l_@@_xdots_shorten_end_dim }
3654 }

```

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

```

3655 \@@_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.

```

3656 \@@_compute_corners:

```

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

```

3657 \@@_adjust_pos_of_blocks_seq:
3658 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3659 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3660 \IfPackageLoadedTF { tikz }
3661 {
3662     \tikzset
3663     {

```

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

```

3664         every-picture / .style =
3665         {
3666             overlay ,
3667             remember-picture ,
3668             name-prefix = \@@_env: -
3669         }
3670     }
3671 }
3672 { }
3673 \cs_set_eq:NN \ialign \@@_old_ialign:
3674 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3675 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3676 \cs_set_eq:NN \OverBrace \@@_OverBrace
3677 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3678 \cs_set_eq:NN \line \@@_line
3679 \g_@@_pre_code_after_tl
3680 \tl_gclear:N \g_@@_pre_code_after_tl

```

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

```

3681 \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).

```

3682 \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`).

```

3683 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3684 { \@@_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:`.

```

3685 \bool_set_true:N \l_@@_in_code_after_bool
3686 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3687 \scan_stop:
3688 \tl_gclear:N \g_nicematrix_code_after_tl
3689 \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.

```

3690 \tl_if_empty:NF \g_@@_pre_code_before_tl
3691 {
3692     \tl_gput_right:Nx \g_@@_aux_tl
3693     {
3694         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3695         { \exp_not:V \g_@@_pre_code_before_tl }
3696     }
3697     \tl_gclear:N \g_@@_pre_code_before_tl
3698 }
3699 \tl_if_empty:NF \g_nicematrix_code_before_tl
3700 {
3701     \tl_gput_right:Nx \g_@@_aux_tl
3702     {
3703         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3704         { \exp_not:V \g_nicematrix_code_before_tl }
3705     }
3706     \tl_gclear:N \g_nicematrix_code_before_tl
3707 }
3708 \str_gclear:N \g_@@_name_env_str

```

```
3709 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹². 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`.

```
3710 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3711 }
```

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`.

```
3712 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3713 { \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).

```
3714 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3715 {
3716   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3717   { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3718 }
```

The following command must *not* be protected.

```
3719 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3720 {
3721   { #1 }
3722   { #2 }
3723   {
3724     \int_compare:nNnTF { #3 } > { 99 }
3725     { \int_use:N \c@iRow }
3726     { #3 }
3727   }
3728   {
3729     \int_compare:nNnTF { #4 } > { 99 }
3730     { \int_use:N \c@jCol }
3731     { #4 }
3732   }
3733   { #5 }
3734 }
```

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).

```
3735 \hook_gput_code:nnn { begindocument } { . }
3736 {
3737   \cs_new_protected:Npx \@@_draw_dotted_lines:
3738   {
3739     \c_@@_pgfortikzpicture_tl
3740     \@@_draw_dotted_lines_i:
3741     \c_@@_endpgfortikzpicture_tl
3742   }
3743 }
```

¹²e.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:`.

```

3744 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3745 {
3746   \pgfrememberpicturepositiononpagetrue
3747   \pgf@relevantforpicturesizefalse
3748   \g_@@_HVdotsfor_lines_tl
3749   \g_@@_Vdots_lines_tl
3750   \g_@@_Ddots_lines_tl
3751   \g_@@_Iddots_lines_tl
3752   \g_@@_Cdots_lines_tl
3753   \g_@@_Ldots_lines_tl
3754 }

3755 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3756 {
3757   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3758   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3759 }

```

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

```

3760 \pgfdeclareshape { @@_diag_node }
3761 {
3762   \savedanchor { \five }
3763   {
3764     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3765     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3766   }
3767   \anchor { 5 } { \five }
3768   \anchor { center } { \pgfpointorigin }
3769 }

```

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).

```

3770 \cs_new_protected:Npn \@@_create_diag_nodes:
3771 {
3772   \pgfpicture
3773   \pgfrememberpicturepositiononpagetrue
3774   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3775   {
3776     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3777     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3778     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3779     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3780     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3781     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3782     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3783     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3784     \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.

```

3785     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3786     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3787     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3788     \str_if_empty:NF \l_@@_name_str
3789     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3790   }

```

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

```

3791   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }

```

```

3792 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3793 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3794 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3795 \pgfcoordinate
3796 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3797 \pgfnodealias
3798 { \@@_env: - last }
3799 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3800 \str_if_empty:NF \l_@@_name_str
3801 {
3802   \pgfnodealias
3803   { \l_@@_name_str - \int_use:N \l_tmpa_int }
3804   { \@@_env: - \int_use:N \l_tmpa_int }
3805   \pgfnodealias
3806   { \l_@@_name_str - last }
3807   { \@@_env: - last }
3808 }
3809 \endpgfpicture
3810 }

```

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.

```

3811 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3812 {

```

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

```

3813 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

3814 \int_set:Nn \l_@@_initial_i_int { #1 }
3815 \int_set:Nn \l_@@_initial_j_int { #2 }
3816 \int_set:Nn \l_@@_final_i_int { #1 }
3817 \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.

```

3818 \bool_set_false:N \l_@@_stop_loop_bool
3819 \bool_do_until:Nn \l_@@_stop_loop_bool
3820 {
3821   \int_add:Nn \l_@@_final_i_int { #3 }
3822   \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3823 \bool_set_false:N \l_@@_final_open_bool
3824 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3825 {
3826   \int_compare:nNnTF { #3 } = 1
3827   { \bool_set_true:N \l_@@_final_open_bool }
3828   {
3829     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3830     { \bool_set_true:N \l_@@_final_open_bool }
3831   }
3832 }
3833 {
3834   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3835   {
3836     \int_compare:nNnT { #4 } = { -1 }
3837     { \bool_set_true:N \l_@@_final_open_bool }
3838   }
3839   {
3840     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3841     {
3842       \int_compare:nNnT { #4 } = 1
3843       { \bool_set_true:N \l_@@_final_open_bool }
3844     }
3845   }
3846 }
3847 \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.

```

3848 {

```

We do a step backwards.

```

3849 \int_sub:Nn \l_@@_final_i_int { #3 }
3850 \int_sub:Nn \l_@@_final_j_int { #4 }
3851 \bool_set_true:N \l_@@_stop_loop_bool
3852 }

```

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`.

```

3853 {
3854   \cs_if_exist:cTF
3855   {
3856     @@ _ dotted _
3857     \int_use:N \l_@@_final_i_int -
3858     \int_use:N \l_@@_final_j_int
3859   }
3860   {
3861     \int_sub:Nn \l_@@_final_i_int { #3 }
3862     \int_sub:Nn \l_@@_final_j_int { #4 }
3863     \bool_set_true:N \l_@@_final_open_bool
3864     \bool_set_true:N \l_@@_stop_loop_bool
3865   }
3866   {
3867     \cs_if_exist:cTF
3868     {
3869       pgf @ sh @ ns @ \@@_env:
3870       - \int_use:N \l_@@_final_i_int

```

```

3871         - \int_use:N \l_@@_final_j_int
3872     }
3873     { \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.

```

3874     {
3875         \cs_set:cpn
3876         {
3877             @@ _ dotted _
3878             \int_use:N \l_@@_final_i_int -
3879             \int_use:N \l_@@_final_j_int
3880         }
3881         { }
3882     }
3883 }
3884 }
3885 }

```

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

```

3886 \bool_set_false:N \l_@@_stop_loop_bool
3887 \bool_do_until:Nn \l_@@_stop_loop_bool
3888 {
3889     \int_sub:Nn \l_@@_initial_i_int { #3 }
3890     \int_sub:Nn \l_@@_initial_j_int { #4 }
3891     \bool_set_false:N \l_@@_initial_open_bool
3892     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3893     {
3894         \int_compare:nNnTF { #3 } = 1
3895         { \bool_set_true:N \l_@@_initial_open_bool }
3896         {
3897             \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3898             { \bool_set_true:N \l_@@_initial_open_bool }
3899         }
3900     }
3901     {
3902         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3903         {
3904             \int_compare:nNnT { #4 } = 1
3905             { \bool_set_true:N \l_@@_initial_open_bool }
3906         }
3907         {
3908             \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3909             {
3910                 \int_compare:nNnT { #4 } = { -1 }
3911                 { \bool_set_true:N \l_@@_initial_open_bool }
3912             }
3913         }
3914     }
3915     \bool_if:NTF \l_@@_initial_open_bool
3916     {
3917         \int_add:Nn \l_@@_initial_i_int { #3 }
3918         \int_add:Nn \l_@@_initial_j_int { #4 }
3919         \bool_set_true:N \l_@@_stop_loop_bool
3920     }
3921     {
3922         \cs_if_exist:cTF
3923         {

```



```

3924         @@ _ dotted _
3925         \int_use:N \l_@@_initial_i_int -
3926         \int_use:N \l_@@_initial_j_int
3927     }
3928     {
3929         \int_add:Nn \l_@@_initial_i_int { #3 }
3930         \int_add:Nn \l_@@_initial_j_int { #4 }
3931         \bool_set_true:N \l_@@_initial_open_bool
3932         \bool_set_true:N \l_@@_stop_loop_bool
3933     }
3934     {
3935         \cs_if_exist:cTF
3936         {
3937             pgf @ sh @ ns @ \@@_env:
3938             - \int_use:N \l_@@_initial_i_int
3939             - \int_use:N \l_@@_initial_j_int
3940         }
3941         { \bool_set_true:N \l_@@_stop_loop_bool }
3942         {
3943             \cs_set:cpn
3944             {
3945                 @@ _ dotted _
3946                 \int_use:N \l_@@_initial_i_int -
3947                 \int_use:N \l_@@_initial_j_int
3948             }
3949             { }
3950         }
3951     }
3952 }
3953 }

```

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.

```

3954     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3955     {
3956         { \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`.

```

3957         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3958         { \int_use:N \l_@@_final_i_int }
3959         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3960         { } % for the name of the block
3961     }
3962 }

```

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).

```

3963 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3964 {
3965     \int_set:Nn \l_@@_row_min_int 1
3966     \int_set:Nn \l_@@_col_min_int 1
3967     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3968     \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`.

```

3969     \seq_map_inline:Nn \g_@@_submatrix_seq
3970     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
3971 }

```

#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.

```

3972 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3973 {
3974   \bool_if:nT
3975   {
3976     \int_compare_p:n { #3 <= #1 }
3977     && \int_compare_p:n { #1 <= #5 }
3978     && \int_compare_p:n { #4 <= #2 }
3979     && \int_compare_p:n { #2 <= #6 }
3980   }
3981   {
3982     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3983     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3984     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3985     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3986   }
3987 }

3988 \cs_new_protected:Npn \@@_set_initial_coords:
3989 {
3990   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3991   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3992 }
3993 \cs_new_protected:Npn \@@_set_final_coords:
3994 {
3995   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3996   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3997 }
3998 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3999 {
4000   \pgfpointanchor
4001   {
4002     \@@_env:
4003     - \int_use:N \l_@@_initial_i_int
4004     - \int_use:N \l_@@_initial_j_int
4005   }
4006   { #1 }
4007   \@@_set_initial_coords:
4008 }
4009 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4010 {
4011   \pgfpointanchor
4012   {
4013     \@@_env:
4014     - \int_use:N \l_@@_final_i_int
4015     - \int_use:N \l_@@_final_j_int
4016   }
4017   { #1 }
4018   \@@_set_final_coords:
4019 }

4020 \cs_new_protected:Npn \@@_open_x_initial_dim:
4021 {
4022   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4023   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4024   {
4025     \cs_if_exist:cT
4026     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4027     {
4028       \pgfpointanchor
4029       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4030       { west }

```

```

4031         \dim_set:Nn \l_@@_x_initial_dim
4032         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4033     }
4034 }

```

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

```

4035     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4036     {
4037         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4038         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4039         \dim_add:Nn \l_@@_x_initial_dim \col@sep
4040     }
4041 }

4042 \cs_new_protected:Npn \@@_open_x_final_dim:
4043 {
4044     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4045     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4046     {
4047         \cs_if_exist:cT
4048         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4049         {
4050             \pgfpointanchor
4051             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4052             { east }
4053             \dim_set:Nn \l_@@_x_final_dim
4054             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4055         }
4056     }

```

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

```

4057     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4058     {
4059         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4060         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4061         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4062     }
4063 }

```

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.

```

4064 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4065 {
4066     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4067     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4068     {
4069         \@@_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.

```

4070     \group_begin:
4071     \int_compare:nNnTF { #1 } = 0
4072     { \color { nicematrix-first-row } }
4073     {

```

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.

```

4074         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4075         { \color { nicematrix-last-row } }
4076     }
4077     \keys_set:nn { NiceMatrix / xdots } { #3 }
4078     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4079     \@@_actually_draw_Ldots:
4080     \group_end:

```

```

4081     }
4082 }

```

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`.

```

4083 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4084 {
4085   \bool_if:NTF \l_@@_initial_open_bool
4086   {
4087     \@@_open_x_initial_dim:
4088     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4089     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4090   }
4091   { \@@_set_initial_coords_from_anchor:n { base-east } }
4092   \bool_if:NTF \l_@@_final_open_bool
4093   {
4094     \@@_open_x_final_dim:
4095     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4096     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4097   }
4098   { \@@_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 (?).

```

4099   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4100   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4101   \@@_draw_line:
4102 }

```

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.

```

4103 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4104 {
4105   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4106   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4107   {
4108     \@@_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.

```

4109   \group_begin:
4110     \int_compare:nNnTF { #1 } = 0
4111     { \color { nicematrix-first-row } }
4112     {

```

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.

```

4113       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4114       { \color { nicematrix-last-row } }
4115     }
4116   \keys_set:nn { NiceMatrix / xdots } { #3 }

```

```

4117         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4118         \@@_actually_draw_Cdots:
4119     \group_end:
4120 }
4121 }

```

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.`

```

4122 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4123 {
4124     \bool_if:NTF \l_@@_initial_open_bool
4125     { \@@_open_x_initial_dim: }
4126     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4127     \bool_if:NTF \l_@@_final_open_bool
4128     { \@@_open_x_final_dim: }
4129     { \@@_set_final_coords_from_anchor:n { mid-west } }
4130     \bool_lazy_and:nnTF
4131     \l_@@_initial_open_bool
4132     \l_@@_final_open_bool
4133     {
4134         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4135         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4136         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4137         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4138         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4139     }
4140     {
4141         \bool_if:NT \l_@@_initial_open_bool
4142         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4143         \bool_if:NT \l_@@_final_open_bool
4144         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4145     }
4146     \@@_draw_line:
4147 }
4148 \cs_new_protected:Npn \@@_open_y_initial_dim:
4149 {
4150     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4151     \dim_set:Nn \l_@@_y_initial_dim
4152     {
4153         \fp_to_dim:n
4154         {
4155             \pgf@y
4156             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4157         }
4158     } % modified 6.13c
4159     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4160     {
4161         \cs_if_exist:cT
4162         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4163         {
4164             \pgfpointanchor
4165             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4166             { north }

```

```

4167         \dim_set:Nn \l_@@_y_initial_dim
4168         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4169     }
4170 }
4171 }
4172 \cs_new_protected:Npn \@@_open_y_final_dim:
4173 {
4174     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4175     \dim_set:Nn \l_@@_y_final_dim
4176     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4177     % modified 6.13c
4178     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4179     {
4180         \cs_if_exist:cT
4181         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4182         {
4183             \pgfpointanchor
4184             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4185             { south }
4186             \dim_set:Nn \l_@@_y_final_dim
4187             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4188         }
4189     }
4190 }

```

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.

```

4191 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4192 {
4193     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4194     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4195     {
4196         \@@_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.

```

4197     \group_begin:
4198     \int_compare:nNnTF { #2 } = 0
4199     { \color { nicematrix-first-col } }
4200     {
4201         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4202         { \color { nicematrix-last-col } }
4203     }
4204     \keys_set:nn { NiceMatrix / xdots } { #3 }
4205     \tl_if_empty:VF \l_@@_xdots_color_tl
4206     { \color { \l_@@_xdots_color_tl } }
4207     \@@_actually_draw_Vdots:
4208     \group_end:
4209 }
4210 }

```

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`.

```
4211 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4212 {
```

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

```
4213   \bool_set_false:N \l_tmpa_bool
```

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

```
4214   \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
4215   {
4216     \@@_set_initial_coords_from_anchor:n { south-west }
4217     \@@_set_final_coords_from_anchor:n { north-west }
4218     \bool_set:Nn \l_tmpa_bool
4219     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4220   }
```

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

```
4221   \bool_if:NTF \l_@@_initial_open_bool
4222     \@@_open_y_initial_dim:
4223     { \@@_set_initial_coords_from_anchor:n { south } }
4224   \bool_if:NTF \l_@@_final_open_bool
4225     \@@_open_y_final_dim:
4226     { \@@_set_final_coords_from_anchor:n { north } }
4227   \bool_if:NTF \l_@@_initial_open_bool
4228   {
4229     \bool_if:NTF \l_@@_final_open_bool
4230     {
4231       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4232       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4233       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4234       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4235       \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4236     }
```

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.

```
4236       \int_compare:nNnT \l_@@_last_col_int > { -2 }
4237       {
4238         \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
4239         {
4240           \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
4241           \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
4242           \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4243           \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
4244         }
4245       }
4246     }
4247   { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
4248 }
4249 {
4250   \bool_if:NTF \l_@@_final_open_bool
4251   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
4252 }
```

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.

```
4253   \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4254   {
4255     \dim_set:Nn \l_@@_x_initial_dim
4256     {
4257       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4258       \l_@@_x_initial_dim \l_@@_x_final_dim
4259     }
4260     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4261   }
```

```

4262     }
4263   }
4264   \@@_draw_line:
4265 }

```

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.

```

4266 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4267 {
4268   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4269   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4270   {
4271     \@@_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.

```

4272     \group_begin:
4273     \keys_set:nn { NiceMatrix / xdots } { #3 }
4274     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4275     \@@_actually_draw_Ddots:
4276     \group_end:
4277   }
4278 }

```

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`.

```

4279 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4280 {
4281   \bool_if:NTF \l_@@_initial_open_bool
4282   {
4283     \@@_open_y_initial_dim:
4284     \@@_open_x_initial_dim:
4285   }
4286   { \@@_set_initial_coords_from_anchor:n { south-east } }
4287   \bool_if:NTF \l_@@_final_open_bool
4288   {
4289     \@@_open_x_final_dim:
4290     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4291   }
4292   { \@@_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.

```

4293   \bool_if:NT \l_@@_parallelize_diags_bool
4294   {
4295     \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).

```

4296     \int_compare:nNnTF \g_@@_ddots_int = 1

```


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

```

4297     {
4298         \dim_gset:Nn \g_@@_delta_x_one_dim
4299         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4300         \dim_gset:Nn \g_@@_delta_y_one_dim
4301         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4302     }

```

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`.

```

4303     {
4304         \dim_set:Nn \l_@@_y_final_dim
4305         {
4306             \l_@@_y_initial_dim +
4307             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4308             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4309         }
4310     }
4311 }
4312 \@@_draw_line:
4313 }

```

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.

```

4314 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4315 {
4316     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4317     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4318     {
4319         \@@_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.

```

4320     \group_begin:
4321     \keys_set:nn { NiceMatrix / xdots } { #3 }
4322     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4323     \@@_actually_draw_Iddots:
4324     \group_end:
4325 }
4326 }

```

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`.

```

4327 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4328 {
4329     \bool_if:NTF \l_@@_initial_open_bool
4330     {
4331         \@@_open_y_initial_dim:
4332         \@@_open_x_initial_dim:

```

```

4333     }
4334     { \@@_set_initial_coords_from_anchor:n { south-west } }
4335     \bool_if:NTF \l_@@_final_open_bool
4336     {
4337         \@@_open_y_final_dim:
4338         \@@_open_x_final_dim:
4339     }
4340     { \@@_set_final_coords_from_anchor:n { north-east } }
4341     \bool_if:NT \l_@@_parallelize_diags_bool
4342     {
4343         \int_gincr:N \g_@@_iddots_int
4344         \int_compare:nNnTF \g_@@_iddots_int = 1
4345         {
4346             \dim_gset:Nn \g_@@_delta_x_two_dim
4347             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4348             \dim_gset:Nn \g_@@_delta_y_two_dim
4349             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4350         }
4351         {
4352             \dim_set:Nn \l_@@_y_final_dim
4353             {
4354                 \l_@@_y_initial_dim +
4355                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4356                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4357             }
4358         }
4359     }
4360     \@@_draw_line:
4361 }

```

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`

```

4362 \cs_new_protected:Npn \@@_draw_line:
4363 {
4364     \pgfrememberpicturepositiononpagetrue
4365     \pgf@relevantforpicturesizefalse
4366     \bool_lazy_or:nnTF
4367     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4368     \l_@@_dotted_bool
4369     \@@_draw_standard_dotted_line:
4370     \@@_draw_unstandard_dotted_line:
4371 }

```

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.

```

4372 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4373 {
4374   \begin { scope }
4375   \@@_draw_unstandard_dotted_line:o
4376   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4377 }

```

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.

```

4378 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4379 {
4380   \@@_draw_unstandard_dotted_line:nVV
4381   { #1 }
4382   \l_@@_xdots_up_tl
4383   \l_@@_xdots_down_tl
4384 }
4385 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following Tikz styles are for both labels (set by the symbols `_` and `^`) of a continuous line with a non-standard style.

```

4386 \hook_gput_code:nnn { begindocument } { . }
4387 {
4388   \IfPackageLoadedTF { tikz }
4389   {
4390     \tikzset
4391     {
4392       @@_node_above / .style = { sloped , above } ,
4393       @@_node_below / .style = { sloped , below }
4394     }
4395   }
4396   { }
4397 }

4398 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
4399 {
4400   \bool_if:NT \l_@@_xdots_h_labels_bool
4401   {
4402     \tikzset
4403     {
4404       @@_node_above / .style = { auto = left } ,
4405       @@_node_below / .style = { auto = right }
4406     }
4407   }
4408   \draw
4409   [
4410     #1 ,
4411     shorten~> = \l_@@_xdots_shorten_end_dim ,
4412     shorten~< = \l_@@_xdots_shorten_start_dim ,
4413   ]
4414   ( \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).

```

4415   -- node [ @@_node_above ] { $ \scriptstyle #2 $ }
4416   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4417   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4418   \end { scope }
4419 }
4420 \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).

```

4421 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4422 {
4423   \bool_lazy_and:nnF
4424     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4425     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4426   {
4427     \pgfscope
4428     \pgftransformshift
4429       {
4430         \pgfpointlineattime { 0.5 }
4431         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4432         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4433       }
4434     \fp_set:Nn \l_tmpa_fp
4435     {
4436       atand
4437       (
4438         \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4439         \l_@@_x_final_dim - \l_@@_x_initial_dim
4440       )
4441     }
4442     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4443     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4444     \pgfnode
4445       { rectangle }
4446       { south }
4447       {
4448         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4449         {
4450           \c_math_toggle_token
4451           \scriptstyle \l_@@_xdots_up_tl
4452           \c_math_toggle_token
4453         }
4454       }
4455     { }
4456     { \pgfusepath { } }
4457     \pgfnode
4458       { rectangle }
4459       { north }
4460       {
4461         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4462         {
4463           \c_math_toggle_token
4464           \scriptstyle \l_@@_xdots_down_tl
4465           \c_math_toggle_token
4466         }
4467       }
4468     { }
4469     { \pgfusepath { } }
4470     \endpgfscope
4471   }
4472   \group_begin:

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4473   \dim_zero_new:N \l_@@_l_dim
4474   \dim_set:Nn \l_@@_l_dim
4475   {
4476     \fp_to_dim:n
4477     {
4478       sqrt
4479       (

```

```

4480      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4481      +
4482      ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4483    )
4484  }
4485 }

```

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.

```

4486   \bool_lazy_or:nnF
4487   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4488   { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4489   \@@_draw_standard_dotted_line_i:
4490 \group_end:
4491 }
4492 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4493 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4494 {

```

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

```

4495   \bool_if:NTF \l_@@_initial_open_bool
4496   {
4497     \bool_if:NTF \l_@@_final_open_bool
4498     {
4499       \int_set:Nn \l_tmpa_int
4500       { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
4501     }
4502     {
4503       \int_set:Nn \l_tmpa_int
4504       {
4505         \dim_ratio:nn
4506         { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
4507         \l_@@_xdots_inter_dim
4508       }
4509     }
4510   }
4511   {
4512     \bool_if:NTF \l_@@_final_open_bool
4513     {
4514       \int_set:Nn \l_tmpa_int
4515       {
4516         \dim_ratio:nn
4517         { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
4518         \l_@@_xdots_inter_dim
4519       }
4520     }
4521     {
4522       \int_set:Nn \l_tmpa_int
4523       {
4524         \dim_ratio:nn
4525         {
4526           \l_@@_l_dim
4527           - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4528         }
4529         \l_@@_xdots_inter_dim
4530       }
4531     }
4532   }

```

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

```

4533 \dim_set:Nn \l_tmpa_dim
4534 {
4535   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4536   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4537 }
4538 \dim_set:Nn \l_tmpb_dim
4539 {
4540   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4541   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4542 }

```

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.

```

4543 \dim_gadd:Nn \l_@@_x_initial_dim
4544 {
4545   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4546   \dim_ratio:nn
4547   {
4548     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4549     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4550   }
4551   { 2 \l_@@_l_dim }
4552 }
4553 \dim_gadd:Nn \l_@@_y_initial_dim
4554 {
4555   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4556   \dim_ratio:nn
4557   {
4558     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4559     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4560   }
4561   { 2 \l_@@_l_dim }
4562 }
4563 \pgf@relevantforpicturesizefalse
4564 \int_step_inline:nnn 0 \l_tmpa_int
4565 {
4566   \pgfpathcircle
4567   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4568   { \l_@@_xdots_radius_dim }
4569   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4570   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4571 }
4572 \pgfusepathqfill
4573 }

```

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.

```

4574 \hook_gput_code:nnn { begindocument } { . }
4575 {
4576   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }

```

```

4577 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4578 \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4579 {
4580   \int_compare:nNnTF \c@jCol = 0
4581   { \@@_error:nn { in~first~col } \Ldots }
4582   {
4583     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4584     { \@@_error:nn { in~last~col } \Ldots }
4585     {
4586       \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4587       { #1 , down = #2 , up = #3 }
4588     }
4589   }
4590   \bool_if:NF \l_@@_nullify_dots_bool
4591   { \phantom { \ensuremath { \@@_old_ldots } } }
4592   \bool_gset_true:N \g_@@_empty_cell_bool
4593 }

4594 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4595 {
4596   \int_compare:nNnTF \c@jCol = 0
4597   { \@@_error:nn { in~first~col } \Cdots }
4598   {
4599     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4600     { \@@_error:nn { in~last~col } \Cdots }
4601     {
4602       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4603       { #1 , down = #2 , up = #3 }
4604     }
4605   }
4606   \bool_if:NF \l_@@_nullify_dots_bool
4607   { \phantom { \ensuremath { \@@_old_cdots } } }
4608   \bool_gset_true:N \g_@@_empty_cell_bool
4609 }

4610 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4611 {
4612   \int_compare:nNnTF \c@iRow = 0
4613   { \@@_error:nn { in~first~row } \Vdots }
4614   {
4615     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4616     { \@@_error:nn { in~last~row } \Vdots }
4617     {
4618       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4619       { #1 , down = #2 , up = #3 }
4620     }
4621   }
4622   \bool_if:NF \l_@@_nullify_dots_bool
4623   { \phantom { \ensuremath { \@@_old_vdots } } }
4624   \bool_gset_true:N \g_@@_empty_cell_bool
4625 }

4626 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4627 {
4628   \int_case:nnF \c@iRow
4629   {
4630     0 { \@@_error:nn { in~first~row } \Ddots }
4631     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4632   }
4633   {
4634     \int_case:nnF \c@jCol

```

```

4635         {
4636             0 { \@@_error:nn { in~first~col } \Ddots }
4637             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4638         }
4639         {
4640             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4641             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4642             { #1 , down = #2 , up = #3 }
4643         }
4644     }
4645 }
4646 \bool_if:NF \l_@@_nullify_dots_bool
4647 { \phantom { \ensuremath { \@@_old_ddots } } }
4648 \bool_gset_true:N \g_@@_empty_cell_bool
4649 }

4650 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4651 {
4652     \int_case:nnF \c@iRow
4653     {
4654         0 { \@@_error:nn { in~first~row } \Iddots }
4655         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4656     }
4657     {
4658         \int_case:nnF \c@jCol
4659         {
4660             0 { \@@_error:nn { in~first~col } \Iddots }
4661             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4662         }
4663         {
4664             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4665             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4666             { #1 , down = #2 , up = #3 }
4667         }
4668     }
4669     \bool_if:NF \l_@@_nullify_dots_bool
4670     { \phantom { \ensuremath { \@@_old_iddots } } }
4671     \bool_gset_true:N \g_@@_empty_cell_bool
4672 }
4673 }

```

End of the \AddToHook.

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

```

4674 \keys_define:nn { NiceMatrix / Ddots }
4675 {
4676     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4677     draw-first .default:n = true ,
4678     draw-first .value_forbidden:n = true
4679 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

4680 \cs_new_protected:Npn \@@_Hspace:
4681 {
4682     \bool_gset_true:N \g_@@_empty_cell_bool
4683     \hspace
4684 }

```

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.

```

4685 \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`.

```

4686 \cs_new:Npn \@@_Hdotsfor:
4687 {
4688   \bool_lazy_and:nnTF
4689     { \int_compare_p:nNn \c@jCol = 0 }
4690     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4691     {
4692       \bool_if:NTF \g_@@_after_col_zero_bool
4693       {
4694         \multicolumn { 1 } { c } { }
4695         \@@_Hdotsfor_i
4696       }
4697       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4698     }
4699   {
4700     \multicolumn { 1 } { c } { }
4701     \@@_Hdotsfor_i
4702   }
4703 }

```

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:`).

```

4704 \hook_gput_code:nnn { begindocument } { . }
4705 {
4706   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4707   \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.

```

4708   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4709   {
4710     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4711     {
4712       \@@_Hdotsfor:nnnn
4713       { \int_use:N \c@iRow }
4714       { \int_use:N \c@jCol }
4715       { #2 }
4716       {
4717         #1 , #3 ,
4718         down = \exp_not:n { #4 } ,
4719         up = \exp_not:n { #5 }
4720       }
4721     }
4722     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4723   }
4724 }

```

```

4725 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4726 {
4727   \bool_set_false:N \l_@@_initial_open_bool
4728   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4729   \int_set:Nn \l_@@_initial_i_int { #1 }
4730   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

4731   \int_compare:nNnTF { #2 } = 1
4732   {
4733     \int_set:Nn \l_@@_initial_j_int 1

```

```

4734     \bool_set_true:N \l_@@_initial_open_bool
4735   }
4736   {
4737     \cs_if_exist:cTF
4738     {
4739       pgf @ sh @ ns @ \@@_env:
4740       - \int_use:N \l_@@_initial_i_int
4741       - \int_eval:n { #2 - 1 }
4742     }
4743     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4744     {
4745       \int_set:Nn \l_@@_initial_j_int { #2 }
4746       \bool_set_true:N \l_@@_initial_open_bool
4747     }
4748   }
4749   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4750   {
4751     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4752     \bool_set_true:N \l_@@_final_open_bool
4753   }
4754   {
4755     \cs_if_exist:cTF
4756     {
4757       pgf @ sh @ ns @ \@@_env:
4758       - \int_use:N \l_@@_final_i_int
4759       - \int_eval:n { #2 + #3 }
4760     }
4761     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4762     {
4763       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4764       \bool_set_true:N \l_@@_final_open_bool
4765     }
4766   }
4767   \group_begin:
4768   \int_compare:nNnTF { #1 } = 0
4769   { \color { nicematrix-first-row } }
4770   {
4771     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4772     { \color { nicematrix-last-row } }
4773   }
4774   \keys_set:nn { NiceMatrix / xdots } { #4 }
4775   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4776   \@@_actually_draw_ldots:
4777   \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).

```

4778   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4779   { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4780   }

4781   \hook_gput_code:nnn { begindocument } { . }
4782   {
4783     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4784     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4785     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4786     {
4787       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4788       {
4789         \@@_Vdotsfor:nnnn
4790         { \int_use:N \c@iRow }

```

```

4791         { \int_use:N \c@jCol }
4792         { #2 }
4793         {
4794             #1 , #3 ,
4795             down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4796         }
4797     }
4798 }
4799 }

```

Enf of \AddToHook.

```

4800 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4801 {
4802     \bool_set_false:N \l_@@_initial_open_bool
4803     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4804     \int_set:Nn \l_@@_initial_j_int { #2 }
4805     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

4806     \int_compare:nNnTF #1 = 1
4807     {
4808         \int_set:Nn \l_@@_initial_i_int 1
4809         \bool_set_true:N \l_@@_initial_open_bool
4810     }
4811     {
4812         \cs_if_exist:cTF
4813         {
4814             pgf @ sh @ ns @ \@@_env:
4815             - \int_eval:n { #1 - 1 }
4816             - \int_use:N \l_@@_initial_j_int
4817         }
4818         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4819         {
4820             \int_set:Nn \l_@@_initial_i_int { #1 }
4821             \bool_set_true:N \l_@@_initial_open_bool
4822         }
4823     }
4824     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4825     {
4826         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4827         \bool_set_true:N \l_@@_final_open_bool
4828     }
4829     {
4830         \cs_if_exist:cTF
4831         {
4832             pgf @ sh @ ns @ \@@_env:
4833             - \int_eval:n { #1 + #3 }
4834             - \int_use:N \l_@@_final_j_int
4835         }
4836         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4837         {
4838             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4839             \bool_set_true:N \l_@@_final_open_bool
4840         }
4841     }
4842     \group_begin:
4843     \int_compare:nNnTF { #2 } = 0
4844     { \color { nicematrix-first-col } }
4845     {
4846         \int_compare:nNnT { #2 } = \g_@@_col_total_int
4847         { \color { nicematrix-last-col } }
4848     }

```

```

4849 \keys_set:nn { NiceMatrix / xdots } { #4 }
4850 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4851 \@@_actually_draw_Vdots:
4852 \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).

```

4853 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4854 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4855 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

4856 \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).¹³

```

4857 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4858 {
4859   \tl_if_empty:nTF { #2 }
4860   { #1 }
4861   { \@@_double_int_eval_i:n #1-#2 \q_stop }
4862 }
4863 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
4864 { \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).

```

4865 \hook_gput_code:nnn { begindocument } { . }
4866 {
4867   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4868   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4869   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4870   {
4871     \group_begin:
4872     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4873     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4874     \use:e
4875     {
4876       \@@_line_i:nn

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

4877         { \@@_double_int_eval:n #2 - \q_stop }
4878         { \@@_double_int_eval:n #3 - \q_stop }
4879     }
4880     \group_end:
4881 }
4882 }
4883 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4884 {
4885     \bool_set_false:N \l_@@_initial_open_bool
4886     \bool_set_false:N \l_@@_final_open_bool
4887     \bool_if:nTF
4888     {
4889         \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4890         ||
4891         \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4892     }
4893     {
4894         \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4895     }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

4896     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
4897 }
4898 \hook_gput_code:nnn { begindocument } { . }
4899 {
4900     \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4901     {

```

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:`.

```

4902     \c_@@_pgfortikzpicture_tl
4903     \@@_draw_line_iii:nn { #1 } { #2 }
4904     \c_@@_endpgfortikzpicture_tl
4905 }
4906 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

4907 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4908 {
4909     \pgfrememberpicturepositiononpagetrue
4910     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4911     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4912     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4913     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4914     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4915     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4916     \@@_draw_line:
4917 }

```

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`

```

4918 \keys_define:nn { NiceMatrix / RowStyle }
4919 {
4920     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,

```

```

4921 cell-space-top-limit .initial:n = \c_zero_dim ,
4922 cell-space-top-limit .value_required:n = true ,
4923 cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4924 cell-space-bottom-limit .initial:n = \c_zero_dim ,
4925 cell-space-bottom-limit .value_required:n = true ,
4926 cell-space-limits .meta:n =
4927 {
4928     cell-space-top-limit = #1 ,
4929     cell-space-bottom-limit = #1 ,
4930 } ,
4931 color .tl_set:N = \l_@@_color_tl ,
4932 color .value_required:n = true ,
4933 bold .bool_set:N = \l_tmpa_bool ,
4934 bold .default:n = true ,
4935 bold .initial:n = false ,
4936 nb-rows .code:n =
4937     \str_if_eq:nnTF { #1 } { * }
4938     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
4939     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
4940 nb-rows .value_required:n = true ,
4941 rowcolor .tl_set:N = \l_tmpa_tl ,
4942 rowcolor .value_required:n = true ,
4943 rowcolor .initial:n = ,
4944 unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
4945 }

```

```

4946 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4947 {
4948     \group_begin:
4949     \tl_clear:N \l_tmpa_tl % value of \rowcolor
4950     \tl_clear:N \l_@@_color_tl
4951     \int_set:Nn \l_@@_key_nb_rows_int 1
4952     \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

4953     \tl_if_empty:NF \l_tmpa_tl
4954     {

```

First, the end of the current row (we remind that \RowStyle applies to the *end* of the current row).

```

4955         \tl_gput_right:Nx \g_@@_pre_code_before_tl
4956         {

```

The command \@@_exp_color_arg:NV is *fully expandable*.

```

4957         \@@_exp_color_arg:NV \@@_rectanglecolor \l_tmpa_tl
4958         { \int_use:N \c@iRow - \int_use:N \c@jCol }
4959         { \int_use:N \c@iRow - * }
4960     }

```

Then, the other rows (if there is several rows).

```

4961         \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4962         {
4963             \tl_gput_right:Nx \g_@@_pre_code_before_tl
4964             {
4965                 \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
4966                 {
4967                     \int_eval:n { \c@iRow + 1 }
4968                     - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4969                 }
4970             }
4971         }
4972     }
4973     \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4974     \tl_gput_right:Nx \g_@@_row_style_tl
4975     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4976     \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
4977 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4978 {
4979   \tl_gput_right:Nx \g_@@_row_style_tl
4980   {
4981     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4982     {
4983       \dim_set:Nn \l_@@_cell_space_top_limit_dim
4984       { \dim_use:N \l_tmpa_dim }
4985     }
4986   }
4987 }

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
4988 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4989 {
4990   \tl_gput_right:Nx \g_@@_row_style_tl
4991   {
4992     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4993     {
4994       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4995       { \dim_use:N \l_tmpb_dim }
4996     }
4997   }
4998 }

\l_@@_color_tl is the value of the key color of \RowStyle.
4999 \tl_if_empty:NF \l_@@_color_tl
5000 {
5001   \tl_gput_right:Nx \g_@@_row_style_tl
5002   {
5003     \mode_leave_vertical:
5004     \@@_color:n { \l_@@_color_tl }
5005   }
5006 }

\l_tmpa_bool is the value of the key bold.
5007 \bool_if:NT \l_tmpa_bool
5008 {
5009   \tl_gput_right:Nn \g_@@_row_style_tl
5010   {
5011     \if_mode_math:
5012       \c_math_toggle_token
5013       \bfseries \boldmath
5014       \c_math_toggle_token
5015     \else:
5016       \bfseries \boldmath
5017     \fi:
5018   }
5019 }
5020 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5021 \group_end:
5022 \g_@@_row_style_tl
5023 \ignorespaces
5024 }

```

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).

```
5025 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5026 {
```

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.

```
5027 \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`.

```
5028 \str_if_in:nnF { #1 } { !! }
5029 {
5030   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5031   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5032 }
5033 \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5034 {
5035   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5036   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5037 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5038 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5039 }
```

```
5040 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5041 \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`).

```
5042 \cs_new_protected:Npn \@@_actually_color:
5043 {
5044   \pgfpicture
5045   \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.

```
5046 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5047 {
5048   \pgfsetcornersarced
5049   {
5050     \pgfpoint
5051     { \l_@@_tab_rounded_corners_dim }
5052     { \l_@@_tab_rounded_corners_dim }
5053   }
```



```

5054 % \end{macrocode}
5055 % Because we want \pkg{nicematrix} compatible with arrays constructed by
5056 % \pkg{array}, the nodes for the rows and columns (that is to say the nodes
5057 % |row-|\textsl{i} and |col-|\textsl{j}|) have not always the expected position,
5058 % that is to say, there is sometimes a slight shifting of something such as
5059 % |\arrayrulewidth|. Now, for the clipping, we have to change slightly the
5060 % position of that clipping whether a rounded rectangle around the array is
5061 % required. That's the point which is tested in the following line.
5062 % \begin{macrocode}
5063 \bool_if:NTF \l_@@_hvlines_bool
5064 {
5065     \pgfpathrectanglecorners
5066     {
5067         \pgfpointadd
5068         { \@@_qpoint:n { row-1 } }
5069         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5070     }
5071     {
5072         \pgfpointadd
5073         {
5074             \@@_qpoint:n
5075             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5076         }
5077         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5078     }
5079 }
5080 {
5081     \pgfpathrectanglecorners
5082     { \@@_qpoint:n { row-1 } }
5083     {
5084         \pgfpointadd
5085         {
5086             \@@_qpoint:n
5087             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5088         }
5089         { \pgfpoint \c_zero_dim \arrayrulewidth }
5090     }
5091 }
5092 \pgfusepath { clip }
5093 }
5094 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5095 {
5096     \begin { pgfscope }
5097         \@@_color_opacity ##2
5098         \use:c { g_@@_color _ ##1 _t1 }
5099         \tl_gclear:c { g_@@_color _ ##1 _t1 }
5100         \pgfusepath { fill }
5101     \end { pgfscope }
5102 }
5103 \endpgfpicture
5104 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5105 \cs_new_protected:Npn \@@_color_opacity
5106 {
5107     \peek_meaning:NTF [
5108         { \@@_color_opacity:w }
5109         { \@@_color_opacity:w [ ] }
5110     }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5111 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5112 {
5113   \tl_clear:N \l_tmpa_tl
5114   \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.
5115   \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5116   \tl_if_empty:NTF \l_tmpb_tl
5117     { \@declaredcolor }
5118     { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5119 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5120 \keys_define:nn { nicematrix / color-opacity }
5121 {
5122   opacity .tl_set:N          = \l_tmpa_tl ,
5123   opacity .value_required:n = true
5124 }

5125 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5126 {
5127   \tl_set:Nn \l_@@_rows_tl { #1 }
5128   \tl_set:Nn \l_@@_cols_tl { #2 }
5129   \@@_cartesian_path:
5130 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5131 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5132 {
5133   \tl_if_blank:nF { #2 }
5134   {
5135     \@@_add_to_colors_seq:xn
5136     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5137     { \@@_cartesian_color:nn { #3 } { - } }
5138   }
5139 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5140 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5141 {
5142   \tl_if_blank:nF { #2 }
5143   {
5144     \@@_add_to_colors_seq:xn
5145     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5146     { \@@_cartesian_color:nn { - } { #3 } }
5147   }
5148 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5149 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5150 {
5151   \tl_if_blank:nF { #2 }
5152   {
5153     \@@_add_to_colors_seq:xn
5154     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5155     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5156   }
5157 }

```

The last argument is the radius of the corners of the rectangle.

```

5158 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5159 {
5160   \tl_if_blank:nF { #2 }
5161   {
5162     \@@_add_to_colors_seq:xn
5163     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5164     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5165   }
5166 }

```

The last argument is the radius of the corners of the rectangle.

```

5167 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5168 {
5169   \@@_cut_on_hyphen:w #1 \q_stop
5170   \tl_clear_new:N \l_@@_tmpc_tl
5171   \tl_clear_new:N \l_@@_tmpd_tl
5172   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5173   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5174   \@@_cut_on_hyphen:w #2 \q_stop
5175   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5176   \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`.

```

5177   \@@_cartesian_path:n { #3 }
5178 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5179 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5180 {
5181   \clist_map_inline:nn { #3 }
5182   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5183 }

```

```

5184 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5185 {
5186   \int_step_inline:nn { \int_use:N \c@iRow }
5187   {
5188     \int_step_inline:nn { \int_use:N \c@jCol }
5189     {
5190       \int_if_even:nTF { ####1 + ##1 }
5191       { \@@_cellcolor [ #1 ] { #2 } }
5192       { \@@_cellcolor [ #1 ] { #3 } }
5193       { ##1 - ####1 }
5194     }
5195   }
5196 }

```

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”.

```

5197 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5198 {
5199   \@@_rectanglecolor [ #1 ] { #2 }
5200   { 1 - 1 }
5201   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5202 }

```

```

5203 \keys_define:nn { NiceMatrix / rowcolors }
5204 {
5205     respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5206     respect-blocks .default:n = true ,
5207     cols .tl_set:N = \l_@@_cols_tl ,
5208     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5209     restart .default:n = true ,
5210     unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5211 }

```

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*.

```

5212 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5213 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5214     \group_begin:
5215     \seq_clear_new:N \l_@@_colors_seq
5216     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5217     \tl_clear_new:N \l_@@_cols_tl
5218     \tl_set:Nn \l_@@_cols_tl { - }
5219     \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).

```

5220     \int_zero_new:N \l_@@_color_int
5221     \int_set:Nn \l_@@_color_int 1
5222     \bool_if:NT \l_@@_respect_blocks_bool
5223     {

```

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`).

```

5224         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5225         \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5226         { \@@_not_in_exterior_p:nnnnn #1 }
5227     }
5228     \pgfpicture
5229     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5230     \clist_map_inline:nn { #2 }
5231     {
5232         \tl_set:Nn \l_tmpa_tl { ##1 }
5233         \tl_if_in:NnTF \l_tmpa_tl { - }
5234         { \@@_cut_on_hyphen:w ##1 \q_stop }
5235         { \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.

```

5236         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5237         \bool_if:NNTF \l_@@_rowcolors_restart_bool
5238         { \int_set:Nn \l_@@_color_int 1 }
5239         { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
5240         \int_zero_new:N \l_@@_tmpc_int
5241         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5242         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5243         {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5244         \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).

```

5245     \bool_if:NT \l_@@_respect_blocks_bool
5246     {
5247         \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5248         { \@@_intersect_our_row_p:nnnnn ###1 }
5249         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5250     }
5251     \tl_set:Nx \l_@@_rows_tl
5252     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5253     \tl_clear_new:N \l_@@_color_tl
5254     \tl_set:Nx \l_@@_color_tl
5255     {
5256         \@@_color_index:n
5257         {
5258             \int_mod:nn
5259             { \l_@@_color_int - 1 }
5260             { \seq_count:N \l_@@_colors_seq }
5261             + 1
5262         }
5263     }
5264     \tl_if_empty:NF \l_@@_color_tl
5265     {
5266         \@@_add_to_colors_seq:xx
5267         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5268         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5269     }
5270     \int_incr:N \l_@@_color_int
5271     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5272 }
5273 }
5274 \endpgfpicture
5275 \group_end:
5276 }

```

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.

```

5277 \cs_new:Npn \@@_color_index:n #1
5278 {
5279     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5280     { \@@_color_index:n { #1 - 1 } }
5281     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5282 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

5283 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
5284 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

```

```

5285 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5286 {
5287     \int_compare:nNnT { #3 } > \l_tmpb_int
5288     { \int_set:Nn \l_tmpb_int { #3 } }
5289 }

```

```

5290 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5291 {
5292     \bool_lazy_or:nnTF

```

```

5293     { \int_compare_p:nNn { #4 } = \c_zero_int }
5294     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5295     \prg_return_false:
5296     \prg_return_true:
5297 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5298 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5299 {
5300   \bool_if:nTF
5301   {
5302     \int_compare_p:n { #1 <= \l_tmpa_int }
5303     &&
5304     \int_compare_p:n { \l_tmpa_int <= #3 }
5305   }
5306   \prg_return_true:
5307   \prg_return_false:
5308 }

```

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).

```

5309 \cs_new_protected:Npn \@@_cartesian_path:n #1
5310 {
5311   \bool_lazy_and:nnT
5312   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5313   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5314   {
5315     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5316     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5317   }

```

We begin the loop over the columns.

```

5318   \clist_map_inline:Nn \l_@@_cols_tl
5319   {
5320     \tl_set:Nn \l_tmpa_tl { ##1 }
5321     \tl_if_in:NnTF \l_tmpa_tl { - }
5322     { \@@_cut_on_hyphen:w ##1 \q_stop }
5323     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5324     \bool_lazy_or:nnT
5325     { \tl_if_blank_p:V \l_tmpa_tl }
5326     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5327     { \tl_set:Nn \l_tmpa_tl { 1 } }
5328     \bool_lazy_or:nnT
5329     { \tl_if_blank_p:V \l_tmpb_tl }
5330     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5331     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5332     \int_compare:nNnT \l_tmpb_tl > \c@jCol
5333     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

\l_@@_tmpc_tl will contain the number of column.

```

5334   \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.

```

5335   \@@_qpoint:n { col - \l_tmpa_tl }
5336   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5337   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5338   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }

```

```

5339 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5340 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5341 \clist_map_inline:Nn \l_@@_rows_tl
5342 {
5343   \tl_set:Nn \l_tmpa_tl { ####1 }
5344   \tl_if_in:NnTF \l_tmpa_tl { - }
5345   { \@@_cut_on_hyphen:w ####1 \q_stop }
5346   { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5347   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5348   \tl_if_empty:NT \l_tmpb_tl
5349   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5350   \int_compare:nNnT \l_tmpb_tl > \c@iRow
5351   { \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.

```

5352 \seq_if_in:NxF \l_@@_corners_cells_seq
5353 { \l_tmpa_tl - \l_@@_tmpc_tl }
5354 {
5355   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5356   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5357   \@@_qpoint:n { row - \l_tmpa_tl }
5358   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5359   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5360   \pgfpathrectanglecorners
5361   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5362   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5363 }
5364 }
5365 }
5366 }

```

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).

```

5367 \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.

```

5368 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5369 {
5370   \clist_set_eq:NN \l_tmpa_clist #1
5371   \clist_clear:N #1
5372   \clist_map_inline:Nn \l_tmpa_clist
5373   {
5374     \tl_set:Nn \l_tmpa_tl { ##1 }
5375     \tl_if_in:NnTF \l_tmpa_tl { - }
5376     { \@@_cut_on_hyphen:w ##1 \q_stop }
5377     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5378     \bool_lazy_or:nnT
5379     { \tl_if_blank_p:V \l_tmpa_tl }
5380     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5381     { \tl_set:Nn \l_tmpa_tl { 1 } }
5382     \bool_lazy_or:nnT
5383     { \tl_if_blank_p:V \l_tmpb_tl }
5384     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5385     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5386     \int_compare:nNnT \l_tmpb_tl > #2
5387     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5388     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5389     { \clist_put_right:Nn #1 { ####1 } }
5390   }
5391 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```
5392 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5393 {
5394   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5395   {
```

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`).

```
5396     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5397     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5398   }
5399   \ignorespaces
5400 }
```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the tabular.

```
5401 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5402 {
5403   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5404   {
5405     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5406     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5407     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5408   }
5409   \ignorespaces
5410 }
```

```
5411 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5412 {
```

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).

```
5413   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5414   {
```

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).

```
5415     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5416     {
5417       \exp_not:N \columncolor [ #1 ]
5418       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5419     }
5420   }
5421 }
```

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`.

```
5422 \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`.

```
5423 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5424 {
5425   \int_compare:nNnTF \l_@@_first_col_int = 0
5426   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5427   {
5428     \int_compare:nNnTF \c@jCol = 0
5429     {
5430       \int_compare:nNnF \c@iRow = { -1 }
5431       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5432     }
5433     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5434   }
5435 }
```

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.

```
5436 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5437 {
5438   \int_compare:nNnF \c@iRow = 0
5439   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
5440 }
```

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.

```
5441 \keys_define:nn { NiceMatrix / Rules }
5442 {
5443   position .int_set:N = \l_@@_position_int ,
5444   position .value_required:n = true ,
5445   start .int_set:N = \l_@@_start_int ,
5446   start .initial:n = 1 ,
5447   end .code:n =
5448     \bool_lazy_or:nnTF
5449     { \tl_if_empty_p:n { #1 } }
5450     { \str_if_eq_p:nn { #1 } { last } }
5451     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5452     { \int_set:Nn \l_@@_end_int { #1 } }
5453 }
```

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.

```

5454 \keys_define:nn { NiceMatrix / RulesBis }
5455 {
5456   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5457   multiplicity .initial:n = 1 ,
5458   dotted .bool_set:N = \l_@@_dotted_bool ,
5459   dotted .initial:n = false ,
5460   dotted .default:n = true ,
5461   color .code:n = \@@_set_CT@arc@:n { #1 } ,
5462   color .value_required:n = true ,
5463   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5464   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.

```

5465   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5466   tikz .value_required:n = true ,
5467   tikz .initial:n = ,
5468   total-width .dim_set:N = \l_@@_rule_width_dim ,
5469   total-width .value_required:n = true ,
5470   width .meta:n = { total-width = #1 } ,
5471   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5472 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5473 \cs_new_protected:Npn \@@_vline:n #1
5474 {

```

The group is for the options.

```

5475   \group_begin:
5476   \int_zero_new:N \l_@@_end_int
5477   \int_set_eq:NN \l_@@_end_int \c_iRow
5478   \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).

```

5479   \int_compare:nNnT \l_@@_position_int < { \c_jCol + 2 }
5480     \@@_vline_i:
5481   \group_end:
5482 }

```

```

5483 \cs_new_protected:Npn \@@_vline_i:
5484 {
5485   \int_zero_new:N \l_@@_local_start_int
5486   \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`.

```

5487   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5488   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5489     \l_tmpa_tl
5490   {

```

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.

```

5491     \bool_gset_true:N \g_tmpa_bool
5492     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5493       { \@@_test_vline_in_block:nnnnn #1 }
5494     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5495       { \@@_test_vline_in_block:nnnnn #1 }
5496     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq

```

```

5497     { \@@_test_vline_in_stroken_block:nnnn ##1 }
5498     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5499     \bool_if:NTF \g_tmpa_bool
5500     {
5501         \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.

```

5502         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5503     }
5504     {
5505         \int_compare:nNnT \l_@@_local_start_int > 0
5506         {
5507             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5508             \@@_vline_ii:
5509             \int_zero:N \l_@@_local_start_int
5510         }
5511     }
5512 }
5513 \int_compare:nNnT \l_@@_local_start_int > 0
5514 {
5515     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5516     \@@_vline_ii:
5517 }
5518 }

```

```

5519 \cs_new_protected:Npn \@@_test_in_corner_v:
5520 {
5521     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5522     {
5523         \seq_if_in:NxT
5524         \l_@@_corners_cells_seq
5525         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5526         { \bool_set_false:N \g_tmpa_bool }
5527     }
5528     {
5529         \seq_if_in:NxT
5530         \l_@@_corners_cells_seq
5531         { \l_tmpa_tl - \l_tmpb_tl }
5532         {
5533             \int_compare:nNnTF \l_tmpb_tl = 1
5534             { \bool_set_false:N \g_tmpa_bool }
5535             {
5536                 \seq_if_in:NxT
5537                 \l_@@_corners_cells_seq
5538                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5539                 { \bool_set_false:N \g_tmpa_bool }
5540             }
5541         }
5542     }
5543 }

```

```

5544 \cs_new_protected:Npn \@@_vline_ii:
5545 {
5546     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5547     \bool_if:NTF \l_@@_dotted_bool
5548     \@@_vline_iv:
5549     {
5550         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5551         \@@_vline_iii:
5552         \@@_vline_v:
5553     }
5554 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

5555 \cs_new_protected:Npn \@@_vline_iii:
5556 {
5557   \pgfpicture
5558   \pgfrememberpicturepositiononpagetrue
5559   \pgf@relevantforpicturesizefalse
5560   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5561   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5562   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5563   \dim_set:Nn \l_tmpb_dim
5564     {
5565       \pgf@x
5566       - 0.5 \l_@@_rule_width_dim
5567       +
5568       ( \arrayrulewidth * \l_@@_multiplicity_int
5569         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5570     }
5571   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5572   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5573   \bool_lazy_all:nT
5574     {
5575       { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5576       { \cs_if_exist_p:N \CT@drsc@ }
5577       { ! \tl_if_blank_p:V \CT@drsc@ }
5578     }
5579     {
5580       \group_begin:
5581       \CT@drsc@
5582       \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5583       \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5584       \dim_set:Nn \l_@@_tmpd_dim
5585         {
5586           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5587           * ( \l_@@_multiplicity_int - 1 )
5588         }
5589       \pgfpathrectanglecorners
5590         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5591         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5592       \pgfusepath { fill }
5593       \group_end:
5594     }
5595   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5596   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5597   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5598     {
5599       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5600       \dim_sub:Nn \l_tmpb_dim \doublerulesep
5601       \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5602       \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5603     }
5604   \CT@arc@
5605   \pgfsetlinewidth { 1.1 \arrayrulewidth }
5606   \pgfsetrectcap
5607   \pgfusepathqstroke
5608   \endpgfpicture
5609 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

5610 \cs_new_protected:Npn \@@_vline_iv:
5611 {
5612   \pgfpicture
5613   \pgfrememberpicturepositiononpagetrue
5614   \pgf@relevantforpicturesizefalse

```

```

5615 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5616 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5617 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5618 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5619 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5620 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5621 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5622 \CT@arc@
5623 \@@_draw_line:
5624 \endpgfpicture
5625 }

```

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`).

```

5626 \cs_new_protected:Npn \@@_vline_v:
5627 {
5628   \begin {tikzpicture }
5629   \pgfrememberpicturepositiononpagetrue
5630   \pgf@relevantforpicturesizefalse
5631   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5632   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5633   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5634   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5635   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5636   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5637   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5638   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5639   ( \l_tmpb_dim , \l_tmpa_dim ) --
5640   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5641   \end { tikzpicture }
5642 }

```

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).

```

5643 \cs_new_protected:Npn \@@_draw_vlines:
5644 {
5645   \int_step_inline:nnn
5646   {
5647     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5648     1 2
5649   }
5650   {
5651     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5652     { \int_eval:n { \c@jCol + 1 } }
5653     \c@jCol
5654   }
5655   {
5656     \tl_if_eq:NnF \l_@@_vlines_clist { all }
5657     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5658     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5659   }
5660 }

```

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}`.

```

5661 \cs_new_protected:Npn \@@_hline:n #1
5662 {

```

The group is for the options.

```

5663 \group_begin:
5664 \int_zero_new:N \l_@@_end_int
5665 \int_set_eq:NN \l_@@_end_int \c@jCol
5666 \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5667 \@@_hline_i:
5668 \group_end:
5669 }

5670 \cs_new_protected:Npn \@@_hline_i:
5671 {
5672   \int_zero_new:N \l_@@_local_start_int
5673   \int_zero_new:N \l_@@_local_end_int

```

$\l_1\text{tmpa_tl}$ is the number of row and $\l_1\text{tmpb_tl}$ the number of column. When we have found a column corresponding to a rule to draw, we note its number in $\l_1\text{@@_tmpc_tl}$.

```

5674 \tl_set:Nx \l_1tmpa_tl { \int_use:N \l_@@_position_int }
5675 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5676   \l_1tmpb_tl
5677   {

```

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 $\backslash\text{Block}$ or a virtual block corresponding to a dotted line, created by $\backslash\text{Cdots}$, $\backslash\text{Vdots}$, etc.), we will set g_tmpa_bool to false and the small horizontal rule won't be drawn.

```

5678   \bool_gset_true:N \g_tmpa_bool
5679   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5680     { \@@_test_hline_in_block:nnnnn ##1 }
5681   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5682     { \@@_test_hline_in_block:nnnnn ##1 }
5683   \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5684     { \@@_test_hline_in_stroken_block:nnnn ##1 }
5685   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5686   \bool_if:NTF \g_tmpa_bool
5687   {
5688     \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. $\l_1\text{@@_local_start_int}$ will be the starting row of the rule that we will have to draw.

```

5689     { \int_set:Nn \l_@@_local_start_int \l_1tmpb_tl }
5690   }
5691   {
5692     \int_compare:nNnT \l_@@_local_start_int > 0
5693     {
5694       \int_set:Nn \l_@@_local_end_int { \l_1tmpb_tl - 1 }
5695       \@@_hline_ii:
5696       \int_zero:N \l_@@_local_start_int
5697     }
5698   }
5699 }
5700 \int_compare:nNnT \l_@@_local_start_int > 0
5701 {
5702   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5703   \@@_hline_ii:
5704 }
5705 }

```

```

5706 \cs_new_protected:Npn \@@_test_in_corner_h:
5707 {
5708   \int_compare:nNnTF \l_1tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5709   {
5710     \seq_if_in:NxT
5711       \l_@@_corners_cells_seq
5712       { \int_eval:n { \l_1tmpa_tl - 1 } - \l_1tmpb_tl }

```

```

5713         { \bool_set_false:N \g_tmpa_bool }
5714     }
5715     {
5716         \seq_if_in:NxT
5717         \l_@@_corners_cells_seq
5718         { \l_tmpa_tl - \l_tmpb_tl }
5719         {
5720             \int_compare:nNnTF \l_tmpa_tl = 1
5721             { \bool_set_false:N \g_tmpa_bool }
5722             {
5723                 \seq_if_in:NxT
5724                 \l_@@_corners_cells_seq
5725                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5726                 { \bool_set_false:N \g_tmpa_bool }
5727             }
5728         }
5729     }
5730 }

5731 \cs_new_protected:Npn \@@_hline_ii:
5732 {
5733     % \bool_set_false:N \l_@@_dotted_bool
5734     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5735     \bool_if:NTF \l_@@_dotted_bool
5736     \@@_hline_iv:
5737     {
5738         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5739         \@@_hline_iii:
5740         \@@_hline_v:
5741     }
5742 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

5743 \cs_new_protected:Npn \@@_hline_iii:
5744 {
5745     \pgfpicture
5746     \pgfrememberpicturepositiononpagetrue
5747     \pgf@relevantforpicturesizefalse
5748     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5749     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5750     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5751     \dim_set:Nn \l_tmpb_dim
5752     {
5753         \pgf@y
5754         - 0.5 \l_@@_rule_width_dim
5755         +
5756         ( \arrayrulewidth * \l_@@_multiplicity_int
5757           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5758     }
5759     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5760     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5761     \bool_lazy_all:nT
5762     {
5763         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5764         { \cs_if_exist_p:N \CT@drsc@ }
5765         { ! \tl_if_blank_p:V \CT@drsc@ }
5766     }
5767     {
5768         \group_begin:
5769         \CT@drsc@
5770         \dim_set:Nn \l_@@_tmpd_dim
5771         {

```

```

5772         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5773         * ( \l_@@_multiplicity_int - 1 )
5774     }
5775     \pgfpathrectanglecorners
5776     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5777     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5778     \pgfusepathqfill
5779     \group_end:
5780 }
5781 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5782 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5783 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5784 {
5785     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5786     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5787     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5788     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5789 }
5790 \CT@arc@
5791 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5792 \pgfsetrectcap
5793 \pgfusepathqstroke
5794 \endpgfpicture
5795 }

```

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
5796 \cs_new_protected:Npn \l_@@_hline_iv:
```

```
5797 {
```

```
5798     \pgfpicture
```

```
5799     \pgfrememberpicturepositiononpagetrue
```

```
5800     \pgf@relevantforpicturesizefalse
```

```
5801     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
```

```
5802     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
```

```
5803     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
```

```
5804     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
```

```
5805     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
```

```
5806     \int_compare:nNnT \l_@@_local_start_int = 1
```

```
5807     {
```

```
5808         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
```

```
5809         \bool_if:NT \g_@@_NiceArray_bool
```

```
5810         { \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.

```
5811     \tl_if_eq:NnF \g_@@_left_delim_tl (
```

```
5812         { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
```



```

5813     }
5814     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5815     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5816     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5817     {
5818         \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5819         \bool_if:NT \g_@@_NiceArray_bool
5820         { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5821         \tl_if_eq:NnF \g_@@_right_delim_tl )
5822         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5823     }
5824     \CT@arc@
5825     \@@_draw_line:
5826     \endpgfpicture
5827 }

```

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`).

```

5828 \cs_new_protected:Npn \@@_hline_v:
5829 {
5830     \begin { tikzpicture }
5831     \pgfrememberpicturepositiononpagetrue
5832     \pgf@relevantforpicturesizefalse
5833     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5834     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5835     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5836     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5837     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5838     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5839     \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5840     \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5841     ( \l_tmpa_dim , \l_tmpb_dim ) --
5842     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5843     \end { tikzpicture }
5844 }

```

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)).

```

5845 \cs_new_protected:Npn \@@_draw_hlines:
5846 {
5847     \int_step_inline:nnn
5848     {
5849         \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5850         1 2
5851     }
5852     {
5853         \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5854         { \int_eval:n { \c@iRow + 1 } }
5855         \c@iRow
5856     }
5857     {
5858         \tl_if_eq:NnF \l_@@_hlines_clist { all }
5859         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5860         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5861     }
5862 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5863 \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.

```

5864 \cs_set:Npn \@@_Hline_i:n #1
5865 {
5866   \peek_remove_spaces:n
5867   {
5868     \peek_meaning:NTF \Hline
5869     { \@@_Hline_ii:nn { #1 + 1 } }
5870     { \@@_Hline_iii:n { #1 } }
5871   }
5872 }
5873 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5874 \cs_set:Npn \@@_Hline_iii:n #1
5875 {
5876   \peek_meaning:NTF [
5877   { \@@_Hline_iv:nw { #1 } }
5878   { \@@_Hline_iv:nw { #1 } [ ] }
5879 }
5880 \cs_set:Npn \@@_Hline_iv:nw #1 [ #2 ]
5881 {
5882   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
5883   \skip_vertical:n { \l_@@_rule_width_dim }
5884   \tl_gput_right:Nx \g_@@_pre_code_after_tl
5885   {
5886     \@@_hline:n
5887     {
5888       multiplicity = #1 ,
5889       position = \int_eval:n { \c@iRow + 1 } ,
5890       total-width = \dim_use:N \l_@@_rule_width_dim ,
5891       #2
5892     }
5893   }
5894   \egroup
5895 }

```

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).

```

5896 \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`).

```

5897 \cs_new_protected:Npn \@@_custom_line:n #1
5898 {
5899   \str_clear_new:N \l_@@_command_str
5900   \str_clear_new:N \l_@@_ccommand_str
5901   \str_clear_new:N \l_@@_letter_str
5902   \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.

```

5903   \bool_lazy_all:nTF
5904   {

```

```

5905     { \str_if_empty_p:N \l_@@_letter_str }
5906     { \str_if_empty_p:N \l_@@_command_str }
5907     { \str_if_empty_p:N \l_@@_ccommand_str }
5908   }
5909   { @@_error:n { No~letter~and~no~command } }
5910   { \exp_args:NV @@_custom_line_i:n \l_@@_other_keys_tl }
5911 }
5912 \keys_define:nn { NiceMatrix / custom-line }
5913 {
5914   letter .str_set:N = \l_@@_letter_str ,
5915   letter .value_required:n = true ,
5916   command .str_set:N = \l_@@_command_str ,
5917   command .value_required:n = true ,
5918   ccommand .str_set:N = \l_@@_ccommand_str ,
5919   ccommand .value_required:n = true ,
5920 }

```

```

5921 \cs_new_protected:Npn @@_custom_line_i:n #1
5922 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

5923   \bool_set_false:N \l_@@_tikz_rule_bool
5924   \bool_set_false:N \l_@@_dotted_rule_bool
5925   \bool_set_false:N \l_@@_color_bool
5926   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
5927   \bool_if:NT \l_@@_tikz_rule_bool
5928   {
5929     \IfPackageLoadedTF { tikz }
5930     { }
5931     { @@_error:n { tikz~in~custom-line~without~tikz } }
5932     \bool_if:NT \l_@@_color_bool
5933     { @@_error:n { color~in~custom-line~with~tikz } }
5934   }
5935   \bool_if:NT
5936   {
5937     \int_compare_p:nNn \l_@@_multiplicity_int > 1
5938     && \l_@@_dotted_rule_bool
5939   }
5940   { @@_error:n { key~multiplicity~with~dotted } }
5941   \str_if_empty:NF \l_@@_letter_str
5942   {
5943     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
5944     { @@_error:n { Several~letters } }
5945     {
5946       \exp_args:NnV \tl_if_in:NnTF
5947       \c_@@_forbidden_letters_str \l_@@_letter_str
5948       { @@_error:n { Forbidden~letter } }
5949       {

```

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.

```

5950       \keys_define:nx { NiceMatrix / ColumnTypes }
5951       {
5952         \l_@@_letter_str .code:n =
5953         { @@_v_custom_line:n { \exp_not:n { #1 } } }
5954       }
5955     }
5956   }
5957 }
5958 \str_if_empty:NF \l_@@_command_str { @@_h_custom_line:n { #1 } }
5959 \str_if_empty:NF \l_@@_ccommand_str { @@_c_custom_line:n { #1 } }
5960 }

```

```
5961 \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.

```
5962 \keys_define:nn { NiceMatrix / custom-line-bis }
5963 {
5964   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5965   multiplicity .initial:n = 1 ,
5966   multiplicity .value_required:n = true ,
5967   color .code:n = \bool_set_true:N \l_@@_color_bool ,
5968   color .value_required:n = true ,
5969   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5970   tikz .value_required:n = true ,
5971   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5972   dotted .value_forbidden:n = true ,
5973   total-width .code:n = { } ,
5974   total-width .value_required:n = true ,
5975   width .code:n = { } ,
5976   width .value_required:n = true ,
5977   sep-color .code:n = { } ,
5978   sep-color .value_required:n = true ,
5979   unknown .code:n = \@@_error:n { Unknown-key-for-custom-line }
5980 }
```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```
5981 \bool_new:N \l_@@_dotted_rule_bool
5982 \bool_new:N \l_@@_tikz_rule_bool
5983 \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`.

```
5984 \keys_define:nn { NiceMatrix / custom-line-width }
5985 {
5986   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5987   multiplicity .initial:n = 1 ,
5988   multiplicity .value_required:n = true ,
5989   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5990   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5991   \bool_set_true:N \l_@@_total_width_bool ,
5992   total-width .value_required:n = true ,
5993   width .meta:n = { total-width = #1 } ,
5994   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5995 }
```

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`).

```
5996 \cs_new_protected:Npn \@@_h_custom_line:n #1
5997 {
```

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`.

```
5998   \cs_set:cpn { nicematrix - \l_@@_command_str }
5999   {
6000     \noalign
6001     {
6002       \@@_compute_rule_width:n { #1 }
6003       \skip_vertical:n { \l_@@_rule_width_dim }
6004       \tl_gput_right:Nx \g_@@_pre_code_after_tl
```

```

6005         {
6006             \@@_hline:n
6007             {
6008                 #1 ,
6009                 position = \int_eval:n { \c@iRow + 1 } ,
6010                 total-width = \dim_use:N \l_@@_rule_width_dim
6011             }
6012         }
6013     }
6014 }
6015 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6016 }
6017 \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`).

```

6018 \cs_new_protected:Npn \@@_c_custom_line:n #1
6019 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6020     \exp_args:Nc \NewExpandableDocumentCommand
6021     { nicematrix - \l_@@_ccommand_str }
6022     { 0 { } m }
6023     {
6024         \noalign
6025         {
6026             \@@_compute_rule_width:n { #1 , ##1 }
6027             \skip_vertical:n { \l_@@_rule_width_dim }
6028             \clist_map_inline:nn
6029             { ##2 }
6030             { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6031         }
6032     }
6033     \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6034 }

```

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*.

```

6035 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6036 {
6037     \str_if_in:nnTF { #2 } { - }
6038     { \@@_cut_on_hyphen:w #2 \q_stop }
6039     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6040     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6041     {
6042         \@@_hline:n
6043         {
6044             #1 ,
6045             start = \l_tmpa_tl ,
6046             end = \l_tmpb_tl ,
6047             position = \int_eval:n { \c@iRow + 1 } ,
6048             total-width = \dim_use:N \l_@@_rule_width_dim
6049         }
6050     }
6051 }
6052 \cs_generate_variant:Nn \@@_c_custom_line:nn { n V }
6053 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6054 {
6055     \bool_set_false:N \l_@@_tikz_rule_bool
6056     \bool_set_false:N \l_@@_total_width_bool
6057     \bool_set_false:N \l_@@_dotted_rule_bool

```

```

6058 \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6059 \bool_if:NF \l_@@_total_width_bool
6060 {
6061   \bool_if:NTF \l_@@_dotted_rule_bool
6062   { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6063   {
6064     \bool_if:NF \l_@@_tikz_rule_bool
6065     {
6066       \dim_set:Nn \l_@@_rule_width_dim
6067       {
6068         \arrayrulewidth * \l_@@_multiplicity_int
6069         + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6070       }
6071     }
6072   }
6073 }
6074 }
6075 \cs_new_protected:Npn \@@_v_custom_line:n #1
6076 {
6077   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6078 \tl_gput_right:Nx \g_@@_preamble_tl
6079 { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6080 \tl_gput_right:Nx \g_@@_pre_code_after_tl
6081 {
6082   \@@_vline:n
6083   {
6084     #1 ,
6085     position = \int_eval:n { \c@jCol + 1 } ,
6086     total-width = \dim_use:N \l_@@_rule_width_dim
6087   }
6088 }
6089 }
6090 \@@_custom_line:n
6091 { 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.

```

6092 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6093 {
6094   \bool_lazy_all:nT
6095   {
6096     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6097     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6098     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6099     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6100   }
6101   { \bool_gset_false:N \g_tmpa_bool }
6102 }

```

The same for vertical rules.

```

6103 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6104 {
6105   \bool_lazy_all:nT
6106   {
6107     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6108     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6109     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }

```

```

6110     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6111   }
6112   { \bool_gset_false:N \g_tmpa_bool }
6113 }
6114 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6115 {
6116   \bool_lazy_all:nT
6117   {
6118     {
6119       ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6120       || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6121     }
6122     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6123     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6124   }
6125   { \bool_gset_false:N \g_tmpa_bool }
6126 }
6127 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6128 {
6129   \bool_lazy_all:nT
6130   {
6131     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6132     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6133     {
6134       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6135       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6136     }
6137   }
6138   { \bool_gset_false:N \g_tmpa_bool }
6139 }

```

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.

```

6140 \cs_new_protected:Npn \@@_compute_corners:
6141 {

```

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.

```

6142   \seq_clear_new:N \l_@@_corners_cells_seq
6143   \clist_map_inline:Nn \l_@@_corners_clist
6144   {
6145     \str_case:nnF { ##1 }
6146     {
6147       { NW }
6148       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6149       { NE }
6150       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6151       { SW }
6152       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6153       { SE }
6154       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6155     }
6156     { \@@_error:nn { bad~corner } { ##1 } }
6157   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```
6158 \seq_if_empty:NF \l_@@_corners_cells_seq
6159 {
```

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.

```
6160 \tl_gput_right:Nx \g_@@_aux_tl
6161 {
6162   \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6163   { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6164 }
6165 }
6166 }
```

“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.

```
6167 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6168 {
```

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.

```
6169 \bool_set_false:N \l_tmpa_bool
6170 \int_zero_new:N \l_@@_last_empty_row_int
6171 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6172 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6173 {
6174   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6175   \bool_lazy_or:nnTF
6176   {
6177     \cs_if_exist_p:c
6178     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6179   }
6180   \l_tmpb_bool
6181   { \bool_set_true:N \l_tmpa_bool }
6182   {
6183     \bool_if:NF \l_tmpa_bool
6184     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6185   }
6186 }
```

Now, you determine the last empty cell in the row of number 1.

```
6187 \bool_set_false:N \l_tmpa_bool
6188 \int_zero_new:N \l_@@_last_empty_column_int
6189 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6190 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6191 {
6192   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6193   \bool_lazy_or:nnTF
6194   \l_tmpb_bool
6195   {
```



```

6196         \cs_if_exist_p:c
6197         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6198     }
6199     { \bool_set_true:N \l_tmpa_bool }
6200     {
6201         \bool_if:NF \l_tmpa_bool
6202         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6203     }
6204 }

```

Now, we loop over the rows.

```

6205     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6206     {

```

We treat the row number ##1 with another loop.

```

6207         \bool_set_false:N \l_tmpa_bool
6208         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6209         {
6210             \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
6211             \bool_lazy_or:nnTF
6212                 \l_tmpb_bool
6213             {
6214                 \cs_if_exist_p:c
6215                 { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
6216             }
6217             { \bool_set_true:N \l_tmpa_bool }
6218             {
6219                 \bool_if:NF \l_tmpa_bool
6220                 {
6221                     \int_set:Nn \l_@@_last_empty_column_int { ####1 }
6222                     \seq_put_right:Nn
6223                         \l_@@_corners_cells_seq
6224                         { ##1 - ####1 }
6225                 }
6226             }
6227         }
6228     }
6229 }

```

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`).

```

6230 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6231 {
6232     \int_set:Nn \l_tmpa_int { #1 }
6233     \int_set:Nn \l_tmpb_int { #2 }
6234     \bool_set_false:N \l_tmpb_bool
6235     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6236     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6237 }
6238 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6239 {
6240     \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6241     {
6242         \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6243         {
6244             \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6245             {
6246                 \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6247                 { \bool_set_true:N \l_tmpb_bool }
6248             }
6249         }
6250     }
6251 }

```

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.

```
6252 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
6253 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6254 {
6255   auto-columns-width .code:n =
6256   {
6257     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6258     \dim_gzero_new:N \g_@@_max_cell_width_dim
6259     \bool_set_true:N \l_@@_auto_columns_width_bool
6260   }
6261 }

6262 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6263 {
6264   \int_gincr:N \g_@@_NiceMatrixBlock_int
6265   \dim_zero:N \l_@@_columns_width_dim
6266   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6267   \bool_if:NT \l_@@_block_auto_columns_width_bool
6268   {
6269     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6270     {
6271       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
6272       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6273     }
6274   }
6275 }
```

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).

```
6276 {
6277   \bool_if:NT \l_@@_block_auto_columns_width_bool
6278   {
6279     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6280     \iow_shipout:Nx \@mainaux
6281     {
6282       \cs_gset:cpn
6283       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6284       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6285     }
6286     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6287   }
6288 }
```

25 The extra nodes

First, two variants of the functions \dim_min:nn and \dim_max:nn.

```
6289 \cs_generate_variant:Nn \dim_min:nn { v n }
6290 \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).

```

6291 \cs_new_protected:Npn \@@_create_extra_nodes:
6292 {
6293   \bool_if:NTF \l_@@_medium_nodes_bool
6294   {
6295     \bool_if:NTF \l_@@_large_nodes_bool
6296     \@@_create_medium_and_large_nodes:
6297     \@@_create_medium_nodes:
6298   }
6299   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6300 }

```

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`.

```

6301 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6302 {
6303   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6304   {
6305     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6306     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6307     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6308     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6309   }
6310   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6311   {
6312     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6313     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6314     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6315     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6316   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6317   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6318   {
6319     \int_step_variable:nnNn
6320     \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.

```

6321     {
6322       \cs_if_exist:cT
6323       { 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`.

```

6324         {
6325             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6326             \dim_set:cn { l_@@_row_\@@_i: _min_dim}
6327             { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6328             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6329             {
6330                 \dim_set:cn { l_@@_column _ \@@_j: _min_dim}
6331                 { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6332             }

```

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`.

```

6333             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6334             \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6335             { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6336             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6337             {
6338                 \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6339                 { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6340             }
6341         }
6342     }
6343 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6344 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6345 {
6346     \dim_compare:nNnT
6347     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6348     {
6349         \@@_qpoint:n { row - \@@_i: - base }
6350         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6351         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6352     }
6353 }
6354 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6355 {
6356     \dim_compare:nNnT
6357     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6358     {
6359         \@@_qpoint:n { col - \@@_j: }
6360         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6361         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6362     }
6363 }
6364 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6365 \cs_new_protected:Npn \@@_create_medium_nodes:
6366 {
6367     \pgfpicture
6368     \pgfrememberpicturepositiononpagetrue
6369     \pgf@relevantforpicturesizefalse
6370     \@@_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”.

```

6371     \tl_set:Nn \l_@@_suffix_tl { -medium }
6372     \@@_create_nodes:

```

```

6373 \endpgfpicture
6374 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁴. 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:`.

```

6375 \cs_new_protected:Npn \@@_create_large_nodes:
6376 {
6377   \pgfpicture
6378   \pgfrememberpicturepositiononpagetrue
6379   \pgf@relevantforpicturesizefalse
6380   \@@_computations_for_medium_nodes:
6381   \@@_computations_for_large_nodes:
6382   \tl_set:Nn \l_@@_suffix_tl { - large }
6383   \@@_create_nodes:
6384   \endpgfpicture
6385 }

6386 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6387 {
6388   \pgfpicture
6389   \pgfrememberpicturepositiononpagetrue
6390   \pgf@relevantforpicturesizefalse
6391   \@@_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”.

```

6392   \tl_set:Nn \l_@@_suffix_tl { - medium }
6393   \@@_create_nodes:
6394   \@@_computations_for_large_nodes:
6395   \tl_set:Nn \l_@@_suffix_tl { - large }
6396   \@@_create_nodes:
6397   \endpgfpicture
6398 }

```

For “large nodes”, the exterior rows and columns don’t interfer. 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.

```

6399 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6400 {
6401   \int_set:Nn \l_@@_first_row_int 1
6402   \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`.

```

6403   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6404   {
6405     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6406     {
6407       (
6408         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6409         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6410       )
6411       / 2
6412     }
6413     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6414     { l_@@_row_\@@_i: _ min_dim }
6415   }
6416   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6417 {
6418   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6419   {
6420     (
6421       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6422       \dim_use:c
6423         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6424     )
6425     / 2
6426   }
6427   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6428   { l_@@_column _ \@@_j: _ max _ dim }
6429 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6430 \dim_sub:cn
6431 { l_@@_column _ 1 _ min _ dim }
6432 \l_@@_left_margin_dim
6433 \dim_add:cn
6434 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6435 \l_@@_right_margin_dim
6436 }

```

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).

```

6437 \cs_new_protected:Npn \@@_create_nodes:
6438 {
6439   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6440   {
6441     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6442     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6443       \@@_pgf_rect_node:nnnnn
6444       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6445       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
6446       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
6447       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
6448       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
6449       \str_if_empty:NF \l_@@_name_str
6450       {
6451         \pgfnodealias
6452         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6453         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6454       }
6455     }
6456   }

```

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 .

```

6457 \cs_if_exist_use:NF
6458   \seq_map_pairwise_function:NNN
6459   \seq_mapthread_function:NNN
6460   \g_@@_multicolumn_cells_seq
6461   \g_@@_multicolumn_sizes_seq
6462   \@@_node_for_multicolumn:nn
6463 }

```

```

6464 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6465 {
6466   \cs_set_nopar:Npn \@@_i: { #1 }
6467   \cs_set_nopar:Npn \@@_j: { #2 }
6468 }

```

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”).

```

6469 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6470 {
6471   \@@_extract_coords_values: #1 \q_stop
6472   \@@_pgf_rect_node:nnnnn
6473     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6474     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6475     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6476     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6477     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6478   \str_if_empty:NF \l_@@_name_str
6479   {
6480     \pgfnodealias
6481       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6482       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6483   }
6484 }

```

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.

```

6485 \keys_define:nn { NiceMatrix / Block / FirstPass }
6486 {
6487   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6488   l .value_forbidden:n = true ,
6489   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6490   r .value_forbidden:n = true ,
6491   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6492   c .value_forbidden:n = true ,
6493   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6494   L .value_forbidden:n = true ,
6495   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6496   R .value_forbidden:n = true ,
6497   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6498   C .value_forbidden:n = true ,
6499   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6500   t .value_forbidden:n = true ,
6501   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6502   b .value_forbidden:n = true ,
6503   color .tl_set:N = \l_@@_color_tl ,
6504   color .value_required:n = true ,
6505   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6506   respect-arraystretch .default:n = true ,
6507 }

```

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.

```
6508 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
6509 {
```

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).

```
6510 \peek_remove_spaces:n
6511 {
6512 \tl_if_blank:nTF { #2 }
6513 { \@@_Block_i 1-1 \q_stop }
6514 {
6515 \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6516 \@@_Block_i_czech \@@_Block_i
6517 #2 \q_stop
6518 }
6519 { #1 } { #3 } { #4 }
6520 }
6521 }
```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
6522 \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`.

```
6523 {
6524 \char_set_catcode_active:N -
6525 \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6526 }
```

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.

```
6527 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6528 {
```

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).

```
6529 \bool_lazy_or:nnTF
6530 { \tl_if_blank_p:n { #1 } }
6531 { \str_if_eq_p:nn { #1 } { * } }
6532 { \int_set:Nn \l_tmpa_int { 100 } }
6533 { \int_set:Nn \l_tmpa_int { #1 } }
6534 \bool_lazy_or:nnTF
6535 { \tl_if_blank_p:n { #2 } }
6536 { \str_if_eq_p:nn { #2 } { * } }
6537 { \int_set:Nn \l_tmpb_int { 100 } }
6538 { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
6539 \int_compare:nNnTF \l_tmpb_int = 1
6540 {
6541 \str_if_empty:NTF \l_@@_hpos_cell_str
6542 { \str_set:Nn \l_@@_hpos_block_str c }
6543 { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6544 }
6545 { \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.

```

6546 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6547 \tl_set:Nx \l_tmpa_tl
6548 {
6549   { \int_use:N \c@iRow }
6550   { \int_use:N \c@jCol }
6551   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6552   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6553 }

```

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).

```

6554 \bool_if:nTF
6555 {
6556   (
6557     \int_compare_p:nNn { \l_tmpa_int } = 1
6558     ||
6559     \int_compare_p:nNn { \l_tmpb_int } = 1
6560   )
6561   && ! \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.

```

6562   && ! \l_@@_X_column_bool
6563 }
6564 { \exp_args:Nxx \@@_Block_iv:nnnnn }
6565 { \exp_args:Nxx \@@_Block_v:nnnnn }
6566 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6567 }

```

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).

```

6568 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6569 {
6570   \int_gincr:N \g_@@_block_box_int
6571   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6572   {
6573     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6574     {
6575       \@@_actually_diagbox:nnnnnn
6576       { \int_use:N \c@iRow }
6577       { \int_use:N \c@jCol }
6578       { \int_eval:n { \c@iRow + #1 - 1 } }
6579       { \int_eval:n { \c@jCol + #2 - 1 } }
6580       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6581     }
6582   }
6583   \box_gclear_new:c
6584   { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }

```

```

6585 \hbox_gset:cn
6586 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6587 {

```

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}`).

```

6588 \tl_if_empty:NTF \l_@@_color_tl
6589 { \int_compare:nNnT { #2 } = 1 \set@color }
6590 { \@@_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`.

```

6591 \int_compare:nNnT { #1 } = 1
6592 {
6593   \int_compare:nNnTF \c@iRow = 0
6594     \l_@@_code_for_first_row_tl
6595     {
6596       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6597         \l_@@_code_for_last_row_tl
6598     }
6599   \g_@@_row_style_tl
6600 }
6601 \group_begin:
6602 \bool_if:NF \l_@@_respect_arraystretch_bool
6603 { \cs_set:Npn \arraystretch { 1 } }
6604 \dim_zero:N \extrarowheight
6605 #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).

```

6606 \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
6607 \bool_if:NTF \l_@@_NiceTabular_bool
6608 {
6609   \bool_lazy_all:nTF
6610   {
6611     { \int_compare_p:nNn { #2 } = 1 }
6612     { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6613     { ! \g_@@_rotate_bool }
6614   }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

6615 {
6616   \use:x
6617   {
6618     \exp_not:N \begin { minipage }%
6619     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6620     { \l_@@_col_width_dim }
6621     \str_case:Vn \l_@@_hpos_block_str
6622     {
6623       c \centering
6624       r \raggedleft
6625       l \raggedright
6626     }
6627   }
6628   #5
6629   \end { minipage }
6630 }
6631 {

```

```

6632         \use:x
6633         {
6634             \exp_not:N \begin { tabular }%
6635             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6636             { @ { } \l_@@_hpos_block_str @ { } }
6637         }
6638         #5
6639         \end { tabular }
6640     }
6641 }
6642 {
6643     \c_math_toggle_token
6644     \use:x
6645     {
6646         \exp_not:N \begin { array }%
6647         [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6648         { @ { } \l_@@_hpos_block_str @ { } }
6649     }
6650     #5
6651     \end { array }
6652     \c_math_toggle_token
6653 }
6654 \group_end:
6655 }
6656 \bool_if:NT \g_@@_rotate_bool
6657 {
6658     \box_grotate:cn
6659     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6660     { 90 }
6661     \bool_gset_false:N \g_@@_rotate_bool
6662 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

6663     \int_compare:nNnT { #2 } = 1
6664     {
6665         \dim_gset:Nn \g_@@_blocks_wd_dim
6666         {
6667             \dim_max:nn
6668             \g_@@_blocks_wd_dim
6669             {
6670                 \box_wd:c
6671                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6672             }
6673         }
6674     }

```

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.

```

6675     \int_compare:nNnT { #1 } = 1
6676     {
6677         \dim_gset:Nn \g_@@_blocks_ht_dim
6678         {
6679             \dim_max:nn
6680             \g_@@_blocks_ht_dim
6681             {
6682                 \box_ht:c
6683                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6684             }
6685         }
6686         \dim_gset:Nn \g_@@_blocks_dp_dim
6687         {
6688             \dim_max:nn
6689             \g_@@_blocks_dp_dim

```

```

6690         {
6691             \box_dp:c
6692             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6693         }
6694     }
6695 }
6696 \seq_gput_right:Nx \g_@@_blocks_seq
6697 {
6698     \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.

```

6699     { \exp_not:n { #3 } , \l_@@_hpos_block_str }
6700     {
6701         \box_use_drop:c
6702         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6703     }
6704 }
6705 }

```

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.

```

6706 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
6707 {
6708     \seq_gput_right:Nx \g_@@_blocks_seq
6709     {
6710         \l_tmpa_tl
6711         { \exp_not:n { #3 } }
6712         {
6713             \bool_if:NTF \l_@@_NiceTabular_bool
6714             {
6715                 \group_begin:
6716                 \bool_if:NF \l_@@_respect_arraystretch_bool
6717                 { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6718                 \exp_not:n
6719                 {
6720                     \dim_zero:N \extrarowheight
6721                     #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).

```

6722             \bool_if:NT \g_@@_rotate_bool
6723             { \str_set:Nn \l_@@_hpos_block_str c }
6724             \use:x
6725             {
6726                 \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
6727                 { @ { } \l_@@_hpos_block_str @ { } }
6728             }
6729             #5
6730             \end { tabular }
6731         }
6732     \group_end:
6733 }
6734 {
6735     \group_begin:
6736     \bool_if:NF \l_@@_respect_arraystretch_bool
6737     { \cs_set:Npn \exp_not:N \arraystretch { 1 } }

```

```

6738         \exp_not:n
6739         {
6740             \dim_zero:N \extrarowheight
6741             #4
6742             \bool_if:NT \g_@@_rotate_bool
6743             { \str_set:Nn \l_@@_hpos_block_str c }
6744             \c_math_toggle_token
6745             \use:x
6746             {
6747                 \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
6748                 { @ { } \l_@@_hpos_block_str @ { } }
6749             }
6750             #5
6751             \end { array }
6752             \c_math_toggle_token
6753         }
6754     \group_end:
6755 }
6756 }
6757 }
6758 }

```

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).

```

6759 \keys_define:nn { NiceMatrix / Block / SecondPass }
6760 {
6761     tikz .code:n =
6762         \IfPackageLoadedTF { tikz }
6763         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6764         { \@@_error:n { tikz-key~without~tikz } } ,
6765     tikz .value_required:n = true ,
6766     fill .code:n =
6767         \tl_set_rescan:Nnn
6768         \l_@@_fill_tl
6769         { \char_set_catcode_other:N ! }
6770         { #1 } ,
6771     fill .value_required:n = true ,
6772     draw .code:n =
6773         \tl_set_rescan:Nnn
6774         \l_@@_draw_tl
6775         { \char_set_catcode_other:N ! }
6776         { #1 } ,
6777     draw .default:n = default ,
6778     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6779     rounded-corners .default:n = 4 pt ,
6780     color .code:n =
6781         \@@_color:n { #1 }
6782         \tl_set_rescan:Nnn
6783         \l_@@_draw_tl
6784         { \char_set_catcode_other:N ! }
6785         { #1 } ,
6786     color .value_required:n = true ,
6787     borders .clist_set:N = \l_@@_borders_clist ,
6788     borders .value_required:n = true ,
6789     hvlines .meta:n = { vlines , hlines } ,
6790     vlines .bool_set:N = \l_@@_vlines_block_bool ,
6791     vlines .default:n = true ,
6792     hlines .bool_set:N = \l_@@_hlines_block_bool ,
6793     hlines .default:n = true ,
6794     line-width .dim_set:N = \l_@@_line_width_dim ,
6795     line-width .value_required:n = true ,

```

```

6796 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6797 l .value_forbidden:n = true ,
6798 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6799 r .value_forbidden:n = true ,
6800 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6801 c .value_forbidden:n = true ,
6802 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6803     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6804 L .value_forbidden:n = true ,
6805 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6806     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6807 R .value_forbidden:n = true ,
6808 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6809     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6810 C .value_forbidden:n = true ,
6811 t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6812 t .value_forbidden:n = true ,
6813 T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6814 T .value_forbidden:n = true ,
6815 b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6816 b .value_forbidden:n = true ,
6817 B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6818 B .value_forbidden:n = true ,
6819 v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
6820 v-center .value_forbidden:n = true ,
6821 name .tl_set:N = \l_@@_block_name_str ,
6822 name .value_required:n = true ,
6823 name .initial:n = ,
6824 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6825 respect-arraystretch .default:n = true ,
6826 transparent .bool_set:N = \l_@@_transparent_bool ,
6827 transparent .default:n = true ,
6828 transparent .initial:n = false ,
6829 unknown .code:n = \@@_error:n { Unknown-key-for-Block }
6830 }

```

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.

```

6831 \cs_new_protected:Npn \@@_draw_blocks:
6832 {
6833   \cs_set_eq:NN \ialign \@@_old_ialign:
6834   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
6835 }
6836 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6837 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6838   \int_zero_new:N \l_@@_last_row_int
6839   \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.

```

6840   \int_compare:nNnTF { #3 } > { 99 }
6841     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
6842     { \int_set:Nn \l_@@_last_row_int { #3 } }
6843   \int_compare:nNnTF { #4 } > { 99 }

```

```

6844 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
6845 { \int_set:Nn \l_@@_last_col_int { #4 } }
6846 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6847 {
6848   \int_compare:nTF
6849   { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6850   {
6851     \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
6852     \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
6853     \@@_msg_redirect_name:nn { columns-not-used } { none }
6854   }
6855   { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6856 }
6857 {
6858   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6859   { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6860   { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6861 }
6862 }

```

#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

```

6863 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6864 {

```

The group is for the keys.

```

6865   \group_begin:
6866   \int_compare:nNnT { #1 } = { #3 }
6867   { \str_set:Nn \l_@@_vpos_of_block_str { t } }
6868   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
6869   \bool_if:NT \l_@@_vlines_block_bool
6870   {
6871     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6872     {
6873       \@@_vlines_block:nnn
6874       { \exp_not:n { #5 } }
6875       { #1 - #2 }
6876       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6877     }
6878   }
6879   \bool_if:NT \l_@@_hlines_block_bool
6880   {
6881     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6882     {
6883       \@@_hlines_block:nnn
6884       { \exp_not:n { #5 } }
6885       { #1 - #2 }
6886       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6887     }
6888   }
6889   \bool_if:nF
6890   {
6891     \l_@@_transparent_bool
6892     || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
6893   }
6894   {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

6895   \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6896   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6897 }

```

```

6898 \bool_lazy_and:nnT
6899 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
6900 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
6901 { \@@_error:n { hlines-with-color } } }

6902 \tl_if_empty:NF \l_@@_draw_tl
6903 {
6904   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6905   {
6906     \@@_stroke_block:nnn
6907     { \exp_not:n { #5 } }
6908     { #1 - #2 }
6909     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6910   }
6911   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6912   { { #1 } { #2 } { #3 } { #4 } }
6913 }

6914 \clist_if_empty:NF \l_@@_borders_clist
6915 {
6916   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6917   {
6918     \@@_stroke_borders_block:nnn
6919     { \exp_not:n { #5 } }
6920     { #1 - #2 }
6921     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6922   }
6923 }

6924 \tl_if_empty:NF \l_@@_fill_tl
6925 {
6926   \tl_gput_right:Nx \g_@@_pre_code_before_tl
6927   {
6928     \exp_not:N \roundedrectanglecolor
6929     \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
6930       { \l_@@_fill_tl }
6931       { { \l_@@_fill_tl } }
6932       { #1 - #2 }
6933       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6934       { \dim_use:N \l_@@_rounded_corners_dim }
6935     ]
6936   }
6937 }

6937 \seq_if_empty:NF \l_@@_tikz_seq
6938 {
6939   \tl_gput_right:Nx \g_nicematrix_code_before_tl
6940   {
6941     \@@_block_tikz:nnnnn
6942     { #1 }
6943     { #2 }
6944     { \int_use:N \l_@@_last_row_int }
6945     { \int_use:N \l_@@_last_col_int }
6946     { \seq_use:Nn \l_@@_tikz_seq { , } }
6947   }
6948 }

6949 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6950 {
6951   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6952   {
6953     \@@_actually_diagbox:nnnnnn
6954     { #1 }
6955     { #2 }
6956     { \int_use:N \l_@@_last_row_int }

```



```

6957         { \int_use:N \l_@@_last_col_int }
6958         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6959     }
6960 }

6961 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6962 \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.

```

6963 \pgfpicture
6964 \pgfrememberpicturepositiononpagetrue
6965 \pgf@relevantforpicturesizefalse
6966 \@@_qpoint:n { row - #1 }
6967 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6968 \@@_qpoint:n { col - #2 }
6969 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6970 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6971 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6972 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6973 \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.

```

6974 \@@_pgf_rect_node:nnnnn
6975 { \@@_env: - #1 - #2 - block }
6976 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6977 \str_if_empty:NF \l_@@_block_name_str
6978 {
6979     \pgfnodealias
6980     { \@@_env: - \l_@@_block_name_str }
6981     { \@@_env: - #1 - #2 - block }
6982     \str_if_empty:NF \l_@@_name_str
6983     {
6984         \pgfnodealias
6985         { \l_@@_name_str - \l_@@_block_name_str }
6986         { \@@_env: - #1 - #2 - block }
6987     }
6988 }

```

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.

```

6989     \bool_if:NF \l_@@_hpos_of_block_cap_bool
6990     {
6991         \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.

```

6992         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6993         {

```

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.

```

6994             \cs_if_exist:cT
6995             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6996             {
6997                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6998                 {
6999                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7000                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7001                 }
7002             }
7003         }

```

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.

```

7004         \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7005         {
7006             \@@_qpoint:n { col - #2 }
7007             \dim_set_eq:NN \l_tmpb_dim \pgf@x
7008         }
7009         \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7010         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7011         {
7012             \cs_if_exist:cT
7013             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7014             {
7015                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7016                 {
7017                     \pgfpointanchor
7018                     { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7019                     { east }
7020                     \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7021                 }
7022             }
7023         }
7024         \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7025         {
7026             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7027             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7028         }
7029         \@@_pgf_rect_node:nnnnn
7030         { \@@_env: - #1 - #2 - block - short }
7031         \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7032     }

```

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.

```

7033     \bool_if:NT \l_@@_medium_nodes_bool
7034     {
7035         \@@_pgf_rect_node:nnn
7036         { \@@_env: - #1 - #2 - block - medium }
7037         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7038         {

```

```

7039         \pgfpointanchor
7040         { \@@_env:
7041           - \int_use:N \l_@@_last_row_int
7042           - \int_use:N \l_@@_last_col_int - medium
7043         }
7044         { south~east }
7045     }
7046 }

```

Now, we will put the label of the block.

```

7047 \bool_lazy_any:nTF
7048 {
7049   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7050   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7051   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7052 }

```

```

7053 {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7054 \int_compare:nNnT { #2 } = 0
7055 { \str_set:Nn \l_@@_hpos_block_str r }
7056 \bool_if:nT \g_@@_last_col_found_bool
7057 {
7058   \int_compare:nNnT { #2 } = \g_@@_col_total_int
7059   { \str_set:Nn \l_@@_hpos_block_str l }
7060 }
7061 \tl_set:Nx \l_tmpa_tl
7062 {
7063   \str_case:Vn \l_@@_vpos_of_block_str
7064   {
7065     c {
7066       \str_case:Vn \l_@@_hpos_block_str
7067       {
7068         c { center }
7069         l { west }
7070         r { east }
7071       }
7072     }
7073     T {
7074       \str_case:Vn \l_@@_hpos_block_str
7075       {
7076         c { north }
7077         l { north~west }
7078         r { north~east }
7079       }
7080     }
7081     B {
7082       \str_case:Vn \l_@@_hpos_block_str
7083       {
7084         c { south }
7085         l { south~west }
7086         r { south~east }
7087       }
7088     }
7089   }
7090 }
7091 }
7092 }
7093 }
7094 \pgftransformshift
7095 {
7096   \pgfpointanchor

```

```

7097         {
7098             \@@_env: - #1 - #2 - block
7099             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7100         }
7101         { \l_tmpa_tl }
7102     }
7103     \pgfset
7104     {
7105         inner~xsep = \c_zero_dim ,
7106         inner~ysep = \l_@@_block_ysep_dim
7107     }
7108     \pgfnode
7109     { rectangle }
7110     { \l_tmpa_tl }
7111     { \box_use_drop:N \l_@@_cell_box } { } { }
7112 }
7113 {
7114     \pgfextracty \l_tmpa_dim
7115     {
7116         \@@_qpoint:n
7117         {
7118             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7119             - base
7120         }
7121     }
7122     \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.

```

7123     \pgfpointanchor
7124     {
7125         \@@_env: - #1 - #2 - block
7126         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7127     }
7128     {
7129         \str_case:Vn \l_@@_hpos_block_str
7130         {
7131             c { center }
7132             l { west }
7133             r { east }
7134         }
7135     }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

7136     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7137     \pgfset { inner~sep = \c_zero_dim }
7138     \pgfnode
7139     { rectangle }
7140     {
7141         \str_case:Vn \l_@@_hpos_block_str
7142         {
7143             c { base }
7144             l { base~west }
7145             r { base~east }
7146         }
7147     }
7148     { \box_use_drop:N \l_@@_cell_box } { } { }
7149 }
7150 \endpgfpicture
7151 \group_end:
7152 }

```

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).

```

7153 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7154 {
7155   \group_begin:
7156   \tl_clear:N \l_@@_draw_tl
7157   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7158   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7159   \pgfpicture
7160   \pgfrememberpicturepositiononpagetrue
7161   \pgf@relevantforpicturesizefalse
7162   \tl_if_empty:NF \l_@@_draw_tl
7163   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7164     \str_if_eq:VnTF \l_@@_draw_tl { default }
7165     { \CT@arc@ }
7166     { \@@_color:V \l_@@_draw_tl }
7167   }
7168   \pgfsetcornersarced
7169   {
7170     \pgfpoint
7171     { \dim_use:N \l_@@_rounded_corners_dim }
7172     { \dim_use:N \l_@@_rounded_corners_dim }
7173   }
7174   \@@_cut_on_hyphen:w #2 \q_stop
7175   \bool_lazy_and:nnT
7176   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7177   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7178   {
7179     \@@_qpoint:n { row - \l_tmpa_tl }
7180     \dim_set:Nn \l_tmpb_dim { \pgf@y }
7181     \@@_qpoint:n { col - \l_tmpb_tl }
7182     \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
7183     \@@_cut_on_hyphen:w #3 \q_stop
7184     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7185     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7186     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7187     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7188     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7189     \dim_set:Nn \l_tmpa_dim { \pgf@y }
7190     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7191     \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
7192     \pgfpathrectanglecorners
7193     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7194     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7195     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7196     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7197     { \pgfusepathqstroke }
7198     { \pgfusepath { stroke } }
7199   }
7200   \endpgfpicture
7201   \group_end:
7202 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7203 \keys_define:nn { NiceMatrix / BlockStroke }
7204 {
7205   color .tl_set:N = \l_@@_draw_tl ,
7206   draw .tl_set:N = \l_@@_draw_tl ,
7207   draw .default:n = default ,
7208   line-width .dim_set:N = \l_@@_line_width_dim ,
7209   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,

```

```

7210     rounded-corners .default:n = 4 pt
7211 }

```

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).

```

7212 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7213 {
7214     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7215     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7216     \@@_cut_on_hyphen:w #2 \q_stop
7217     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7218     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7219     \@@_cut_on_hyphen:w #3 \q_stop
7220     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7221     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7222     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7223     {
7224         \use:x
7225         {
7226             \@@_vline:n
7227             {
7228                 position = ##1 ,
7229                 start = \l_@@_tmpc_tl ,
7230                 end = \int_eval:n { \l_tmpa_tl - 1 } ,
7231                 total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7232             }
7233         }
7234     }
7235 }
7236 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7237 {
7238     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7239     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7240     \@@_cut_on_hyphen:w #2 \q_stop
7241     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7242     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7243     \@@_cut_on_hyphen:w #3 \q_stop
7244     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7245     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7246     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7247     {
7248         \use:x
7249         {
7250             \@@_hline:n
7251             {
7252                 position = ##1 ,
7253                 start = \l_@@_tmpd_tl ,
7254                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
7255                 total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7256             }
7257         }
7258     }
7259 }

```

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).

```

7260 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7261 {
7262     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7263     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }

```

```

7264 \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7265 { \@@_error:n { borders~forbidden } }
7266 {
7267   \tl_clear_new:N \l_@@_borders_tikz_tl
7268   \keys_set:nV
7269     { NiceMatrix / OnlyForTikzInBorders }
7270   \l_@@_borders_clist
7271   \@@_cut_on_hyphen:w #2 \q_stop
7272   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7273   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7274   \@@_cut_on_hyphen:w #3 \q_stop
7275   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7276   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7277   \@@_stroke_borders_block_i:
7278 }
7279 }
7280 \hook_gput_code:nnn { begindocument } { . }
7281 {
7282   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7283   {
7284     \c_@@_pgfortikzpicture_tl
7285     \@@_stroke_borders_block_ii:
7286     \c_@@_endpgfortikzpicture_tl
7287   }
7288 }
7289 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7290 {
7291   \pgfrememberpicturepositiononpagetrue
7292   \pgf@relevantforpicturesizefalse
7293   \CT@arc@
7294   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7295   \clist_if_in:NnT \l_@@_borders_clist { right }
7296     { \@@_stroke_vertical:n \l_tmpb_tl }
7297   \clist_if_in:NnT \l_@@_borders_clist { left }
7298     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7299   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7300     { \@@_stroke_horizontal:n \l_tmpa_tl }
7301   \clist_if_in:NnT \l_@@_borders_clist { top }
7302     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7303 }
7304 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7305 {
7306   tikz .code:n =
7307     \cs_if_exist:NTF \tikzpicture
7308       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7309       { \@@_error:n { tikz-in~borders~without-tikz } } ,
7310   tikz .value_required:n = true ,
7311   top .code:n = ,
7312   bottom .code:n = ,
7313   left .code:n = ,
7314   right .code:n = ,
7315   unknown .code:n = \@@_error:n { bad~border }
7316 }

```

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).

```

7317 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7318 {
7319   \@@_qpoint:n \l_@@_tmpc_tl
7320   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7321   \@@_qpoint:n \l_tmpa_tl
7322   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }

```

```

7323 \@@_qpoint:n { #1 }
7324 \tl_if_empty:NTF \l_@@_borders_tikz_tl
7325 {
7326   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7327   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7328   \pgfusepathqstroke
7329 }
7330 {
7331   \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7332   ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7333 }
7334 }

```

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).

```

7335 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7336 {
7337   \@@_qpoint:n \l_@@_tmpd_tl
7338   \clist_if_in:NnTF \l_@@_borders_clist { left }
7339   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7340   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7341   \@@_qpoint:n \l_tmpb_tl
7342   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7343   \@@_qpoint:n { #1 }
7344   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7345   {
7346     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7347     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7348     \pgfusepathqstroke
7349   }
7350   {
7351     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7352     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7353   }
7354 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

7355 \keys_define:nn { NiceMatrix / BlockBorders }
7356 {
7357   borders .clist_set:N = \l_@@_borders_clist ,
7358   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7359   rounded-corners .default:n = 4 pt ,
7360   line-width .dim_set:N = \l_@@_line_width_dim ,
7361 }

```

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.

```

7362 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7363 {
7364   \begin { tikzpicture }
7365   \clist_map_inline:nn { #5 }
7366   {
7367     \path [ ##1 ]
7368     ( #1 -| #2 )
7369     rectangle
7370     ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7371   }
7372   \end { tikzpicture }
7373 }

```


27 How to draw the dotted lines transparently

```

7374 \cs_set_protected:Npn \@@_renew_matrix:
7375 {
7376   \RenewDocumentEnvironment { pmatrix } { }
7377   { \pNiceMatrix }
7378   { \endpNiceMatrix }
7379   \RenewDocumentEnvironment { vmatrix } { }
7380   { \vNiceMatrix }
7381   { \endvNiceMatrix }
7382   \RenewDocumentEnvironment { Vmatrix } { }
7383   { \VNiceMatrix }
7384   { \endVNiceMatrix }
7385   \RenewDocumentEnvironment { bmatrix } { }
7386   { \bNiceMatrix }
7387   { \endbNiceMatrix }
7388   \RenewDocumentEnvironment { Bmatrix } { }
7389   { \BNiceMatrix }
7390   { \endBNiceMatrix }
7391 }

```

28 Automatic arrays

We will extract the potential keys `columns-type`, `l`, `c`, `r` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7392 \keys_define:nn { NiceMatrix / Auto }
7393 {
7394   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7395   columns-type .value_required:n = true ,
7396   l .meta:n = { columns-type = l } ,
7397   r .meta:n = { columns-type = r } ,
7398   c .meta:n = { columns-type = c } ,
7399   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7400   delimiters / color .value_required:n = true ,
7401   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7402   delimiters / max-width .default:n = true ,
7403   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7404   delimiters .value_required:n = true ,
7405 }
7406 \NewDocumentCommand \AutoNiceMatrixWithDelims
7407 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7408 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7409 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7410 {

```

The group is for the protection of the keys.

```

7411 \group_begin:
7412 \bool_set_true:N \l_@@_Matrix_bool
7413 \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`).

```

7414 \cs_set_eq:NN \@@_transform_preamble: \prg_do_nothing:
7415 \use:x
7416 {
7417   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7418   { * { #4 } { \exp_not:N \l_@@_columns_type_tl } }
7419   [ \exp_not:N \l_tmpa_tl ]
7420 }
7421 \int_compare:nNnT \l_@@_first_row_int = 0

```

```

7422 {
7423   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7424   \prg_replicate:nn { #4 - 1 } { & }
7425   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7426 }
7427 \prg_replicate:nn { #3 }
7428 {
7429   \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).

```

7430   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7431   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7432 }
7433 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7434 {
7435   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7436   \prg_replicate:nn { #4 - 1 } { & }
7437   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7438 }
7439 \end { NiceArrayWithDelims }
7440 \group_end:
7441 }
7442 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7443 {
7444   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7445   {
7446     \bool_gset_false:N \g_@@_NiceArray_bool
7447     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7448     \AutoNiceMatrixWithDelims { #2 } { #3 }
7449   }
7450 }
7451 \@@_define_com:nnn p ( )
7452 \@@_define_com:nnn b [ ]
7453 \@@_define_com:nnn v | |
7454 \@@_define_com:nnn V \| \|
7455 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

7456 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7457 {
7458   \group_begin:
7459   \bool_gset_true:N \g_@@_NiceArray_bool
7460   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7461   \group_end:
7462 }

```

29 The redefinition of the command \dotfill

```

7463 \cs_set_eq:NN \@@_old_dotfill \dotfill
7464 \cs_new_protected:Npn \@@_dotfill:
7465 {

```

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}).

```

7466   \@@_old_dotfill
7467   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:

```

```
7468 }
```

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 its 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`.

```
7469 \cs_new_protected:Npn \@@_dotfill_i:
7470 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }
```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```
7471 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7472 {
7473   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7474   {
7475     \@@_actually_diagbox:nnnnnn
7476     { \int_use:N \c@iRow }
7477     { \int_use:N \c@jCol }
7478     { \int_use:N \c@iRow }
7479     { \int_use:N \c@jCol }
7480     { \exp_not:n { #1 } }
7481     { \exp_not:n { #2 } }
7482   }
```

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.

```
7483 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7484 {
7485   { \int_use:N \c@iRow }
7486   { \int_use:N \c@jCol }
7487   { \int_use:N \c@iRow }
7488   { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
7489   { }
7490 }
7491 }
```

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.

```
7492 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7493 {
7494   \pgfpicture
7495   \pgf@relevantforpicturesizefalse
7496   \pgfrememberpicturepositiononpagetrue
7497   \@@_qpoint:n { row - #1 }
7498   \dim_set_eq:NN \l_tmpa_dim \pgf@y
7499   \@@_qpoint:n { col - #2 }
7500   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7501   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7502   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7503   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7504   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7505   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7506   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7507   {
```

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.

```

7508     \CT@arc@
7509     \pgfsetroundcap
7510     \pgfusepathqstroke
7511 }
7512 \pgfset { inner~sep = 1 pt }
7513 \pgfscope
7514 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7515 \pgfnode { rectangle } { south-west }
7516 {
7517     \begin { minipage } { 20 cm }
7518     \@@_math_toggle_token: #5 \@@_math_toggle_token:
7519     \end { minipage }
7520 }
7521 { }
7522 { }
7523 \endpgfscope
7524 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7525 \pgfnode { rectangle } { north-east }
7526 {
7527     \begin { minipage } { 20 cm }
7528     \raggedleft
7529     \@@_math_toggle_token: #6 \@@_math_toggle_token:
7530     \end { minipage }
7531 }
7532 { }
7533 { }
7534 \endpgfpicture
7535 }
```

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.

```

7536 \keys_define:nn { NiceMatrix }
7537 {
7538     CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7539     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7540 }
7541 \keys_define:nn { NiceMatrix / CodeAfter }
7542 {
7543     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7544     sub-matrix .value_required:n = true ,
7545     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7546     delimiters / color .value_required:n = true ,
7547     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7548     rules .value_required:n = true ,
7549     unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
7550 }
```

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. 78.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

7551 \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 `\`.

```
7552 \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`.

```
7553 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7554 {
7555   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7556   \@@_CodeAfter_iv:n
7557 }
```

We catch the argument of the command `\end` (in `#1`).

```
7558 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7559 {
```

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.

```
7560   \str_if_eq:eeTF \currentenv { #1 }
7561   { \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`.

```
7562   {
7563     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7564     \@@_CodeAfter_ii:n
7565   }
7566 }
```

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.

```
7567 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7568 {
7569   \pgfpicture
7570   \pgfrememberpicturepositiononpagetrue
7571   \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.

```
7572   \@@_qpoint:n { row - 1 }
7573   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7574   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7575   \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).

```
7576   \bool_if:nTF { #3 }
7577   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7578   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
```

```

7579 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7580 {
7581   \cs_if_exist:cT
7582   { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
7583   {
7584     \pgfpointanchor
7585     { \l_@@_env: - ##1 - #2 }
7586     { \bool_if:nTF { #3 } { west } { east } }
7587     \dim_set:Nn \l_tmpa_dim
7588     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7589   }
7590 }

```

Now we can put the delimiter with a node of PGF.

```

7591 \pgfset { inner~sep = \c_zero_dim }
7592 \dim_zero:N \nulldelimiterspace
7593 \pgftransformshift
7594 {
7595   \pgfpoint
7596   { \l_tmpa_dim }
7597   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7598 }
7599 \pgfnode
7600 { rectangle }
7601 { \bool_if:nTF { #3 } { east } { west } }
7602 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7603 \nullfont
7604 \c_math_toggle_token
7605 \l_@@_color:V \l_@@_delimiters_color_tl
7606 \bool_if:nTF { #3 } { \left #1 } { \left . }
7607 \vcenter
7608 {
7609   \nullfont
7610   \hrule \@height
7611   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7612   \@depth \c_zero_dim
7613   \@width \c_zero_dim
7614 }
7615 \bool_if:nTF { #3 } { \right . } { \right #1 }
7616 \c_math_toggle_token
7617 }
7618 { }
7619 { }
7620 \endpgfpicture
7621 }

```

33 The command `\SubMatrix`

```

7622 \keys_define:nn { NiceMatrix / sub-matrix }
7623 {
7624   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7625   extra-height .value_required:n = true ,
7626   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7627   left-xshift .value_required:n = true ,
7628   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7629   right-xshift .value_required:n = true ,
7630   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7631   xshift .value_required:n = true ,
7632   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

7633   delimiters / color .value_required:n = true ,
7634   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7635   slim .default:n = true ,
7636   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7637   hlines .default:n = all ,
7638   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7639   vlines .default:n = all ,
7640   hvlines .meta:n = { hlines, vlines } ,
7641   hvlines .value_forbidden:n = true ,
7642 }
7643 \keys_define:nn { NiceMatrix }
7644 {
7645   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7646   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7647   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7648   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7649   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7650   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7651 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

7652 \keys_define:nn { NiceMatrix / SubMatrix }
7653 {
7654   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7655   delimiters / color .value_required:n = true ,
7656   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7657   hlines .default:n = all ,
7658   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7659   vlines .default:n = all ,
7660   hvlines .meta:n = { hlines, vlines } ,
7661   hvlines .value_forbidden:n = true ,
7662   name .code:n =
7663     \tl_if_empty:nTF { #1 }
7664     { \@@_error:n { Invalid-name } }
7665     {
7666       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7667       {
7668         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7669         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7670         {
7671           \str_set:Nn \l_@@_submatrix_name_str { #1 }
7672           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7673         }
7674       }
7675       { \@@_error:n { Invalid-name } }
7676     } ,
7677   name .value_required:n = true ,
7678   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7679   rules .value_required:n = true ,
7680   code .tl_set:N = \l_@@_code_tl ,
7681   code .value_required:n = true ,
7682   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
7683 }

7684 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7685 {
7686   \peek_remove_spaces:n
7687   {
7688     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7689     {
7690       \SubMatrix { #1 } { #2 } { #3 } { #4 }
7691       [

```

```

7692         delimiters / color = \l_@@_delimiters_color_tl ,
7693         hlines = \l_@@_submatrix_hlines_clist ,
7694         vlines = \l_@@_submatrix_vlines_clist ,
7695         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
7696         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
7697         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
7698         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
7699         #5
7700     ]
7701 }
7702 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7703 }
7704 }
7705 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
7706 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7707 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
7708 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
7709 {
7710     \seq_gput_right:Nx \g_@@_submatrix_seq
7711     {
We use \str_if_eq:nnTF because it is fully expandable.
7712         { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
7713         { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
7714         { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
7715         { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
7716     }
7717 }

```

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.

```

7718 \hook_gput_code:nnn { begindocument } { . }
7719 {
7720     \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
7721     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
7722     \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
7723     {
7724         \peek_remove_spaces:n
7725         {
7726             \@@_sub_matrix:nnnnnnn
7727             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
7728         }
7729     }
7730 }

```


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).

```

7731 \NewDocumentCommand \@@_compute_i_j:nn
7732 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7733 { \@@_compute_i_j:nnnn #1 #2 }

7734 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
7735 {
7736   \tl_set:Nn \l_@@_first_i_tl { #1 }
7737   \tl_set:Nn \l_@@_first_j_tl { #2 }
7738   \tl_set:Nn \l_@@_last_i_tl { #3 }
7739   \tl_set:Nn \l_@@_last_j_tl { #4 }
7740   \tl_if_eq:NnT \l_@@_first_i_tl { last }
7741     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
7742   \tl_if_eq:NnT \l_@@_first_j_tl { last }
7743     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
7744   \tl_if_eq:NnT \l_@@_last_i_tl { last }
7745     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
7746   \tl_if_eq:NnT \l_@@_last_j_tl { last }
7747     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
7748 }

7749 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
7750 {
7751   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

7752   \@@_compute_i_j:nn { #2 } { #3 }
7753   % added 6.19b
7754   \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
7755     { \cs_set:Npn \arraystretch { 1 } }
7756   \bool_lazy_or:nnTF
7757     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7758     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7759     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
7760   {
7761     \str_clear_new:N \l_@@_submatrix_name_str
7762     \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
7763     \pgfpicture
7764     \pgfrememberpicturerepositiononpagetrue
7765     \pgf@relevantforpicturesizefalse
7766     \pgfset { inner~sep = \c_zero_dim }
7767     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7768     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

7769   \bool_if:NTF \l_@@_submatrix_slim_bool
7770     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
7771     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
7772     {
7773       \cs_if_exist:cT
7774         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7775         {
7776           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7777           \dim_set:Nn \l_@@_x_initial_dim
7778             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7779         }
7780       \cs_if_exist:cT
7781         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7782         {
7783           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7784           \dim_set:Nn \l_@@_x_final_dim
7785             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7786         }

```

```

7787     }
7788     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7789     { \@@_error:nn { Impossible~delimiter } { left } }
7790     {
7791         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7792         { \@@_error:nn { Impossible~delimiter } { right } }
7793         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7794     }
7795     \endpgfpicture
7796 }
7797 \group_end:
7798 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7799 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7800 {
7801     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7802     \dim_set:Nn \l_@@_y_initial_dim
7803     {
7804         \fp_to_dim:n
7805         {
7806             \pgf@y
7807             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
7808         }
7809     } % modified 6.13c
7810     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7811     \dim_set:Nn \l_@@_y_final_dim
7812     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
7813     % modified 6.13c
7814     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7815     {
7816         \cs_if_exist:cT
7817         { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7818         {
7819             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7820             \dim_set:Nn \l_@@_y_initial_dim
7821             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7822         }
7823         \cs_if_exist:cT
7824         { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7825         {
7826             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7827             \dim_set:Nn \l_@@_y_final_dim
7828             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7829         }
7830     }
7831     \dim_set:Nn \l_tmpa_dim
7832     {
7833         \l_@@_y_initial_dim - \l_@@_y_final_dim +
7834         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7835     }
7836     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7837     \group_begin:
7838     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7839     \@@_set_CT@arc@:V \l_@@_rules_color_tl
7840     \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`.

```

7841 \seq_map_inline:Nn \g_@@_cols_vlism_seq
7842 {
7843   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7844   {
7845     \int_compare:nNnT
7846       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
7847     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

7848       \@@_qpoint:n { col - ##1 }
7849       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7850       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7851       \pgfusepathqstroke
7852     }
7853   }
7854 }

```

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.

```

7855 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7856 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7857 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7858 {
7859   \bool_lazy_and:nnTF
7860     { \int_compare_p:nNn { ##1 } > 0 }
7861     {
7862       \int_compare_p:nNn
7863         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7864     {
7865       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7866       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7867       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7868       \pgfusepathqstroke
7869     }
7870     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
7871 }

```

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.

```

7872 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7873 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7874 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7875 {
7876   \bool_lazy_and:nnTF
7877     { \int_compare_p:nNn { ##1 } > 0 }
7878     {
7879       \int_compare_p:nNn
7880         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7881     {
7882       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7883 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

7884 \dim_set:Nn \l_tmpa_dim
7885 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7886 \str_case:nn { #1 }
7887 {
7888   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7889   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7890   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7891   }
7892   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

7893         \dim_set:Nn \l_tmpb_dim
7894         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7895         \str_case:nn { #2 }
7896         {
7897             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7898             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7899             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7900         }
7901         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7902         \pgfusepathqstroke
7903         \group_end:
7904     }
7905     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7906 }

```

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).

```

7907     \str_if_empty:NF \l_@@_submatrix_name_str
7908     {
7909         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7910         \l_@@_x_initial_dim \l_@@_y_initial_dim
7911         \l_@@_x_final_dim \l_@@_y_final_dim
7912     }
7913     \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`.

```

7914     \begin { pgfscope }
7915     \pgftransformshift
7916     {
7917         \pgfpoint
7918         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7919         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7920     }
7921     \str_if_empty:NTF \l_@@_submatrix_name_str
7922     { \@@_node_left:nn #1 { } }
7923     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7924     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7925     \pgftransformshift
7926     {
7927         \pgfpoint
7928         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7929         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7930     }
7931     \str_if_empty:NTF \l_@@_submatrix_name_str
7932     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7933     {
7934         \@@_node_right:nnnn #2
7935         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7936     }
7937     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7938     \flag_clear_new:n { nicematrix }
7939     \l_@@_code_tl
7940 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms i - j , row - i , col - j and i - $|j$ refer to the

number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```
7941 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor
```

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 `\@@_pgfpntanchor_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.

```
7942 \cs_new_protected:Npn \@@_pgfpntanchor:n #1
7943 {
7944   \use:e
7945   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
7946 }
```

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 -.

```
7947 \cs_new:Npn \@@_pgfpntanchor_i:nn #1 #2
7948 { #1 { \@@_pgfpntanchor_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.

```
7949 \tl_const:Nn \c_@@_integers_alist_tl
7950 {
7951   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7952   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7953   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7954   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7955 }
```

```
7956 \cs_new:Npn \@@_pgfpntanchor_ii:w #1-#2\q_stop
7957 {
```

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`.

```
7958   \tl_if_empty:nTF { #2 }
7959   {
7960     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7961     {
7962       \flag_raise:n { nicematrix }
7963       \int_if_even:nTF { \flag_height:n { nicematrix } }
7964       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7965       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7966     }
7967     { #1 }
7968   }
```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```
7969   { \@@_pgfpntanchor_iii:w { #1 } #2 }
7970 }
```

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. `\@@_pgfpntanchor_i:nn`).

```
7971 \cs_new:Npn \@@_pgfpntanchor_iii:w #1 #2 -
```

```

7972 {
7973   \str_case:nnF { #1 }
7974   {
7975     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7976     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7977   }

```

Now the case of a node of the form $i-j$.

```

7978 {
7979   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7980   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7981 }
7982 }

```

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`).

```

7983 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7984 {
7985   \pgfnode
7986   { rectangle }
7987   { east }
7988   {
7989     \nullfont
7990     \c_math_toggle_token
7991     \@@_color:V \l_@@_delimiters_color_tl
7992     \left #1
7993     \vcenter
7994     {
7995       \nullfont
7996       \hrule \@height \l_tmpa_dim
7997         \@depth \c_zero_dim
7998         \@width \c_zero_dim
7999     }
8000     \right .
8001     \c_math_toggle_token
8002   }
8003   { #2 }
8004   { }
8005 }

```

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.

```

8006 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8007 {
8008   \pgfnode
8009   { rectangle }
8010   { west }
8011   {
8012     \nullfont
8013     \c_math_toggle_token
8014     \@@_color:V \l_@@_delimiters_color_tl
8015     \left .
8016     \vcenter
8017     {
8018       \nullfont
8019       \hrule \@height \l_tmpa_dim
8020         \@depth \c_zero_dim
8021         \@width \c_zero_dim
8022     }
8023     \right #1
8024     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }

```

```

8025      ^ { \smash { #4 } }
8026      \c_math_toggle_token
8027    }
8028    { #2 }
8029    { }
8030  }

```

34 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

8031 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8032 {
8033   \peek_remove_spaces:n
8034   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8035 }

8036 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8037 {
8038   \peek_remove_spaces:n
8039   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8040 }

8041 \keys_define:nn { NiceMatrix / Brace }
8042 {
8043   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8044   left-shorten .default:n = true ,
8045   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8046   shorten .meta:n = { left-shorten , right-shorten } ,
8047   right-shorten .default:n = true ,
8048   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8049   yshift .value_required:n = true ,
8050   yshift .initial:n = \c_zero_dim ,
8051   color .tl_set:N = \l_tmpa_tl ,
8052   color .value_required:n = true ,
8053   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8054 }

```

#1 is the first cell of the rectangle (with the syntax $i-j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to `under` or `over`.

```

8055 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8056 {
8057   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8058 \@@_compute_i_j:nn { #1 } { #2 }
8059 \bool_lazy_or:nnTF
8060 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8061 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8062 {
8063   \str_if_eq:nnTF { #5 } { under }
8064   { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8065   { \@@_error:nn { Construct~too~large } { \OverBrace } }
8066 }
8067 {
8068   \tl_clear:N \l_tmpa_tl
8069   \keys_set:nn { NiceMatrix / Brace } { #4 }
8070   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8071   \pgfpicture

```

```

8072 \pgfrememberpicturepositiononpagetrue
8073 \pgf@relevantforpicturesizefalse
8074 \bool_if:NT \l_@@_brace_left_shorten_bool
8075 {
8076   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8077   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8078   {
8079     \cs_if_exist:cT
8080     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8081     {
8082       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8083       \dim_set:Nn \l_@@_x_initial_dim
8084       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8085     }
8086   }
8087 }
8088 \bool_lazy_or:nnT
8089 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8090 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8091 {
8092   \@@_qpoint:n { col - \l_@@_first_j_tl }
8093   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8094 }
8095 \bool_if:NT \l_@@_brace_right_shorten_bool
8096 {
8097   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8098   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8099   {
8100     \cs_if_exist:cT
8101     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8102     {
8103       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8104       \dim_set:Nn \l_@@_x_final_dim
8105       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8106     }
8107   }
8108 }
8109 \bool_lazy_or:nnT
8110 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8111 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8112 {
8113   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8114   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8115 }
8116 \pgfset { inner-sep = \c_zero_dim }
8117 \str_if_eq:nnTF { #5 } { under }
8118 { \@@_underbrace_i:n { #3 } }
8119 { \@@_overbrace_i:n { #3 } }
8120 \endpgfpicture
8121 }
8122 \group_end:
8123 }

```

The argument is the text to put above the brace.

```

8124 \cs_new_protected:Npn \@@_overbrace_i:n #1
8125 {
8126   \@@_qpoint:n { row - \l_@@_first_i_tl }
8127   \pgftransformshift
8128   {
8129     \pgfpoint
8130     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8131     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8132   }
8133   \pgfnode

```



```

8134 { rectangle }
8135 { south }
8136 {
8137   \vbox_top:n
8138   {
8139     \group_begin:
8140     \everycr { }
8141     \halign
8142     {
8143       \hfil ## \hfil \crrc
8144       \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8145       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8146       \c_math_toggle_token
8147       \overbrace
8148       {
8149         \hbox_to_wd:nn
8150         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8151         { }
8152       }
8153       \c_math_toggle_token
8154       \cr
8155     }
8156     \group_end:
8157   }
8158 }
8159 { }
8160 { }
8161 }

```

The argument is the text to put under the brace.

```

8162 \cs_new_protected:Npn \@@_underbrace_i:n #1
8163 {
8164   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8165   \pgftransformshift
8166   {
8167     \pgfpoint
8168     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8169     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8170   }
8171   \pgfnode
8172   { rectangle }
8173   { north }
8174   {
8175     \group_begin:
8176     \everycr { }
8177     \vbox:n
8178     {
8179       \halign
8180       {
8181         \hfil ## \hfil \crrc
8182         \c_math_toggle_token
8183         \underbrace
8184         {
8185           \hbox_to_wd:nn
8186           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8187           { }
8188         }
8189         \c_math_toggle_token
8190         \cr
8191         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8192         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8193       }
8194     }
8195     \group_end:

```

```

8196     }
8197     { }
8198     { }
8199 }

```

35 The command \ShowCellNames

```

8200 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8201 {
8202   \dim_zero_new:N \g_@@_tmpc_dim
8203   \dim_zero_new:N \g_@@_tmpd_dim
8204   \dim_zero_new:N \g_@@_tmpe_dim
8205   \int_step_inline:nn \c@iRow
8206   {
8207     \begin { pgfpicture }
8208     \@@_qpoint:n { row - ##1 }
8209     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8210     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8211     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8212     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8213     \bool_if:NTF \l_@@_in_code_after_bool
8214     \end { pgfpicture }
8215     \int_step_inline:nn \c@jCol
8216     {
8217       \hbox_set:Nn \l_tmpa_box
8218       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8219       \begin { pgfpicture }
8220       \@@_qpoint:n { col - #####1 }
8221       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8222       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8223       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8224       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8225       \endpgfpicture
8226       \end { pgfpicture }
8227       \fp_set:Nn \l_tmpa_fp
8228       {
8229         \fp_min:nn
8230         {
8231           \fp_min:nn
8232           { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8233           { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8234         }
8235         { 1.0 }
8236       }
8237       \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8238       \pgfpicture
8239       \pgfrememberpicturepositiononpagetrue
8240       \pgf@relevantforpicturesizefalse
8241       \pgftransformshift
8242       {
8243         \pgfpoint
8244         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8245         { \dim_use:N \g_tmpa_dim }
8246       }
8247       \pgfnode
8248       { rectangle }
8249       { center }
8250       { \box_use:N \l_tmpa_box }
8251       { }

```

```

8252         { }
8253     \endpgfpicture
8254 }
8255 }
8256 }
8257 \NewDocumentCommand \@@_ShowCellNames { }
8258 {
8259     \bool_if:NT \l_@@_in_code_after_bool
8260     {
8261         \pgfpicture
8262         \pgfrememberpicturepositiononpagetrue
8263         \pgf@relevantforpicturesizefalse
8264         \pgfpathrectanglecorners
8265         { \@@_qpoint:n { 1 } }
8266         {
8267             \@@_qpoint:n
8268             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8269         }
8270         \pgfsetfillopacity { 0.75 }
8271         \pgfsetfillcolor { white }
8272         \pgfusepathqfill
8273         \endpgfpicture
8274     }
8275     \dim_zero_new:N \g_@@_tmpc_dim
8276     \dim_zero_new:N \g_@@_tmpd_dim
8277     \dim_zero_new:N \g_@@_tmpe_dim
8278     \int_step_inline:nn \c@iRow
8279     {
8280         \bool_if:NTF \l_@@_in_code_after_bool
8281         {
8282             \pgfpicture
8283             \pgfrememberpicturepositiononpagetrue
8284             \pgf@relevantforpicturesizefalse
8285         }
8286         { \begin { pgfpicture } }
8287         \@@_qpoint:n { row - ##1 }
8288         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8289         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8290         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8291         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8292         \bool_if:NTF \l_@@_in_code_after_bool
8293         { \endpgfpicture }
8294         { \end { pgfpicture } }
8295         \int_step_inline:nn \c@jCol
8296         {
8297             \hbox_set:Nn \l_tmpa_box
8298             {
8299                 \normalfont \Large \sffamily \bfseries
8300                 \bool_if:NTF \l_@@_in_code_after_bool
8301                 { \color { red } }
8302                 { \color { red ! 50 } }
8303                 ##1 - ####1
8304             }
8305             \bool_if:NTF \l_@@_in_code_after_bool
8306             {
8307                 \pgfpicture
8308                 \pgfrememberpicturepositiononpagetrue
8309                 \pgf@relevantforpicturesizefalse
8310             }
8311             { \begin { pgfpicture } }
8312             \@@_qpoint:n { col - ####1 }
8313             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8314             \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }

```

```

8315 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8316 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8317 \bool_if:NTF \l_@@_in_code_after_bool
8318 { \endpgfpicture }
8319 { \end { pgfpicture } }
8320 \fp_set:Nn \l_tmpa_fp
8321 {
8322   \fp_min:nn
8323   {
8324     \fp_min:nn
8325     { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8326     { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8327   }
8328   { 1.0 }
8329 }
8330 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8331 \pgfpicture
8332 \pgfrememberpicturepositiononpagetrue
8333 \pgf@relevantforpicturesizefalse
8334 \pgftransformshift
8335 {
8336   \pgfpoint
8337   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8338   { \dim_use:N \g_tmpa_dim }
8339 }
8340 \pgfnode
8341 { rectangle }
8342 { center }
8343 { \box_use:N \l_tmpa_box }
8344 { }
8345 { }
8346 \endpgfpicture
8347 }
8348 }
8349 }

```

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.

```

8350 \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.

```

8351 \bool_new:N \c_@@_footnote_bool
8352 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
8353 {
8354   The~key~'\l_keys_key_str'~is~unknown. \\
8355   That~key~will~be~ignored. \\
8356   For~a~list~of~the~available~keys,~type~H~<return>.
8357 }
8358 {
8359   The~available~keys~are~(in~alphabetic~order):~
8360   footnote,~
8361   footnotehyper,~
8362   messages-for-Overleaf,~

```

```

8363     no-test-for-array,~
8364     renew-dots,~and
8365     renew-matrix.
8366 }

8367 \keys_define:nn { NiceMatrix / Package }
8368 {
8369     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8370     renew-dots .value_forbidden:n = true ,
8371     renew-matrix .code:n = \@@_renew_matrix: ,
8372     renew-matrix .value_forbidden:n = true ,
8373     messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
8374     footnote .bool_set:N = \c_@@_footnote_bool ,
8375     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
8376     no-test-for-array .bool_set:N = \c_@@_no_test_for_array_bool ,
8377     no-test-for-array .default:n = true ,
8378     unknown .code:n = \@@_error:n { Unknown-key-for-package }
8379 }
8380 \ProcessKeysOptions { NiceMatrix / Package }

8381 \@@_msg_new:nn { footnote-with-footnotehyper-package }
8382 {
8383     You~can't~use~the~option~'footnote'~because~the~package~
8384     footnotehyper~has~already~been~loaded.~
8385     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8386     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8387     of~the~package~footnotehyper.\\
8388     The~package~footnote~won't~be~loaded.
8389 }

8390 \@@_msg_new:nn { footnotehyper-with-footnote-package }
8391 {
8392     You~can't~use~the~option~'footnotehyper'~because~the~package~
8393     footnote~has~already~been~loaded.~
8394     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8395     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8396     of~the~package~footnote.\\
8397     The~package~footnotehyper~won't~be~loaded.
8398 }

8399 \bool_if:NT \c_@@_footnote_bool
8400 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8401     \IfClassLoadedTF { beamer }
8402     { \bool_set_false:N \c_@@_footnote_bool }
8403     {
8404         \IfPackageLoadedTF { footnotehyper }
8405         { \@@_error:n { footnote-with-footnotehyper-package } }
8406         { \usepackage { footnote } }
8407     }
8408 }

8409 \bool_if:NT \c_@@_footnotehyper_bool
8410 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8411     \IfClassLoadedTF { beamer }
8412     { \bool_set_false:N \c_@@_footnote_bool }
8413     {
8414         \IfPackageLoadedTF { footnote }
8415         { \@@_error:n { footnotehyper-with-footnote-package } }

```

```

8416         { \usepackage { footnotehyper } }
8417     }
8418     \bool_set_true:N \c_@@_footnote_bool
8419 }

```

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.

```

8420 \bool_new:N \l_@@_underscore_loaded_bool
8421 \IfPackageLoadedTF { underscore }
8422 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8423 { }
8424 \hook_gput_code:nnn { begindocument } { . }
8425 {
8426     \bool_if:NF \l_@@_underscore_loaded_bool
8427     {
8428         \IfPackageLoadedTF { underscore }
8429         { \@@_error:n { underscore~after~nicematrix } }
8430         { }
8431     }
8432 }

```

38 Error messages of the package

```

8433 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
8434 { \str_const:Nn \c_@@_available_keys_str { } }
8435 {
8436     \str_const:Nn \c_@@_available_keys_str
8437     { For~a~list~of~the~available~keys,~type~H~<return>. }
8438 }
8439 \seq_new:N \g_@@_types_of_matrix_seq
8440 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8441 {
8442     NiceMatrix ,
8443     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8444 }
8445 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8446 { \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`.

```

8447 \cs_new_protected:Npn \@@_error_too_much_cols:
8448 {
8449     \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8450     {
8451         \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8452         { \@@_fatal:n { too-much-cols-for-matrix } }

```

```

8453     {
8454         \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8455         { \@@_fatal:n { too-much-cols-for-matrix } }
8456         {
8457             \bool_if:NF \l_@@_last_col_without_value_bool
8458             { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
8459         }
8460     }
8461 }
8462 {
8463     \IfPackageLoadedTF { tabularx }
8464     {
8465         \str_if_eq:VnTF \g_@@_name_env_str { NiceTabularX }
8466         {
8467             \int_compare:nNnTF \c@iRow = \c_zero_int
8468             { \@@_fatal:n { X-columns-with-tabularx } }
8469             {
8470                 \@@_fatal:nn { too-much-cols-for-array }
8471                 {
8472                     However,~this-message-may-be-erroneous:~
8473                     maybe-you-have-used~X-columns-while~'tabularx'~is~loaded,~
8474                     ~which-is-forbidden~(however,~it's~still~possible~to~use~
8475                     X-columns-in~{NiceTabularX}).
8476                 }
8477             }
8478         }
8479         { \@@_fatal:nn { too-much-cols-for-array } { } }
8480     }
8481     { \@@_fatal:nn { too-much-cols-for-array } { } }
8482 }
8483 }

```

The following command must *not* be protected since it's used in an error message.

```

8484 \cs_new:Npn \@@_message_hdotsfor:
8485 {
8486     \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8487     { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
8488 }
8489 \@@_msg_new:nn { negative-weight }
8490 {
8491     Negative-weight.\
8492     The-weight-of-the~'X'~columns-must-be-positive-and-you-have-used~
8493     the-value~'\int_use:N \l_@@_weight_int'~.\
8494     The-absolute-value-will-be-used.
8495 }
8496 \@@_msg_new:nn { last-col-not-used }
8497 {
8498     Column-not-used.\
8499     The-key~'last-col'~is~in-force-but-you-have-not-used-that-last-column~
8500     in-your~\@@_full_name_env:~.~However,~you-can-go-on.
8501 }
8502 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
8503 {
8504     Too-much-columns.\
8505     In-the-row~\int_eval:n { \c@iRow },~
8506     you-try-to-use-more-columns~
8507     than-allowed-by-your~\@@_full_name_env:~\@@_message_hdotsfor:\
8508     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
8509     (plus-the-exterior-columns).~This-error-is-fatal.
8510 }
8511 \@@_msg_new:nn { too-much-cols-for-matrix }
8512 {
8513     Too-much-columns.\

```

```

8514   In~the~row~\int_eval:n { \c@iRow },~
8515   you~try~to~use~more~columns~than~allowed~by~your~
8516   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
8517   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
8518   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
8519   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
8520   \token_to_str:N \setcounter\ to~change~that~value).~
8521   This~error~is~fatal.
8522 }

8523 \@@_msg_new:nn { too-much-cols-for-array }
8524 {
8525   Too~much~columns.\\
8526   In~the~row~\int_eval:n { \c@iRow },~
8527   ~you~try~to~use~more~columns~than~allowed~by~your~
8528   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
8529   \int_use:N \g_@@_static_num_of_col_int\
8530   ~(plus~the~potential~exterior~ones).~#1
8531   This~error~is~fatal.
8532 }

8533 \@@_msg_new:nn { X-columns-with-tabularx }
8534 {
8535   There~is~a~problem.\\
8536   You~have~probably~used~X~columns~in~your~environment~{\g_@@_name_env_str}.~
8537   That's~not~allowed~because~'tabularx'~is~loaded~(however,~you~can~use~X~columns~
8538   in~an~environment~{NiceTabularX}).\\
8539   This~error~is~fatal.
8540 }

8541 \@@_msg_new:nn { columns-not-used }
8542 {
8543   Columns~not~used.\\
8544   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8545   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
8546   The~columns~you~did~not~used~won't~be~created.\\
8547   We~won't~have~similar~error~till~the~end~of~the~document.
8548 }

8549 \@@_msg_new:nn { in-first-col }
8550 {
8551   Erroneous~use.\\
8552   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
8553   That~command~will~be~ignored.
8554 }

8555 \@@_msg_new:nn { in-last-col }
8556 {
8557   Erroneous~use.\\
8558   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
8559   That~command~will~be~ignored.
8560 }

8561 \@@_msg_new:nn { in-first-row }
8562 {
8563   Erroneous~use.\\
8564   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
8565   That~command~will~be~ignored.
8566 }

8567 \@@_msg_new:nn { in-last-row }
8568 {
8569   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
8570   That~command~will~be~ignored.
8571 }

8572 \@@_msg_new:nn { caption-outside-float }

```



```

8573 {
8574   Key~caption~forbidden.\\
8575   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
8576   environment.~This~key~will~be~ignored.
8577 }
8578 \@@_msg_new:nn { short-caption~without~caption }
8579 {
8580   You~should~not~use~the~key~'short-caption'~without~'caption'.~
8581   However,~your~'short-caption'~will~be~used~as~'caption'.
8582 }
8583 \@@_msg_new:nn { double-closing-delimiter }
8584 {
8585   Double~delimiter.\\
8586   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
8587   delimiter.~This~delimiter~will~be~ignored.
8588 }
8589 \@@_msg_new:nn { delimiter~after~opening }
8590 {
8591   Double~delimiter.\\
8592   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
8593   delimiter.~That~delimiter~will~be~ignored.
8594 }
8595 \@@_msg_new:nn { bad-option~for~line-style }
8596 {
8597   Bad~line~style.\\
8598   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
8599   is~'standard'.~That~key~will~be~ignored.
8600 }
8601 \@@_msg_new:nn { Identical~notes~in~caption }
8602 {
8603   Identical~tabular~notes.\\
8604   You~can't~put~several~notes~with~the~same~content~in~
8605   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
8606   If~you~go~on,~the~output~will~probably~be~erroneous.
8607 }
8608 \@@_msg_new:nn { tabularnote~below~the~tabular }
8609 {
8610   \token_to_str:N \tabularnote\ forbidden\\
8611   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8612   of~your~tabular~because~the~caption~will~be~composed~below~
8613   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8614   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
8615   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8616   no~similar~error~will~raised~in~this~document.
8617 }
8618 \@@_msg_new:nn { Unknown~key~for~rules }
8619 {
8620   Unknown~key.\\
8621   There~is~only~two~keys~available~here:~width~and~color.\\
8622   You~key~'\l_keys_key_str'~will~be~ignored.
8623 }
8624 \@@_msg_new:nnn { Unknown~key~for~custom~line }
8625 {
8626   Unknown~key.\\
8627   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
8628   It~you~go~on,~you~will~probably~have~other~errors. \\
8629   \c_@@_available_keys_str
8630 }
8631 {
8632   The~available~keys~are~(in~alphabetic~order):~

```

```

8633     ccommand,~
8634     color,~
8635     command,~
8636     dotted,~
8637     letter,~
8638     multiplicity,~
8639     sep-color,~
8640     tikz,~and~total-width.
8641 }
8642 \@@_msg_new:nnn { Unknown~key~for~xdots }
8643 {
8644     Unknown~key.\\
8645     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
8646     \c_@@_available_keys_str
8647 }
8648 {
8649     The~available~keys~are~(in~alphabetic~order):~
8650     'color',~
8651     'horizontal-labels',~
8652     'inter',~
8653     'line-style',~
8654     'radius',~
8655     'shorten',~
8656     'shorten-end'~and~'shorten-start'.
8657 }
8658 \@@_msg_new:nn { Unknown~key~for~rowcolors }
8659 {
8660     Unknown~key.\\
8661     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
8662     (and~you~try~to~use~'\l_keys_key_str')\\
8663     That~key~will~be~ignored.
8664 }
8665 \@@_msg_new:nn { label~without~caption }
8666 {
8667     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
8668     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
8669 }
8670 \@@_msg_new:nn { W~warning }
8671 {
8672     Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
8673     (row~\int_use:N \c@iRow).
8674 }
8675 \@@_msg_new:nn { Construct~too~large }
8676 {
8677     Construct~too~large.\\
8678     Your~command~\token_to_str:N #1
8679     can't~be~drawn~because~your~matrix~is~too~small.\\
8680     That~command~will~be~ignored.
8681 }
8682 \@@_msg_new:nn { underscore~after~nicematrix }
8683 {
8684     Problem~with~'underscore'.\\
8685     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
8686     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
8687     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
8688 }
8689 \@@_msg_new:nn { ampersand~in~light-syntax }
8690 {
8691     Ampersand~forbidden.\\
8692     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
8693     the~key~'light-syntax'~is~in~force.~This~error~is~fatal.

```

```

8694     }
8695 \@@_msg_new:nn { double-backslash-in~light-syntax }
8696 {
8697     Double~backslash~forbidden.\\
8698     You~can't~use~\token_to_str:N
8699     \\~to~separate~rows~because~the~key~'light-syntax'~
8700     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
8701     (set~by~the~key~'end-of-row')~.~This~error~is~fatal.
8702 }
8703 \@@_msg_new:nn { hlines-with-color }
8704 {
8705     Incompatible~keys.\\
8706     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
8707     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
8708     Maybe~it~will~possible~in~future~version.\\
8709     Your~key~will~be~discarded.
8710 }
8711 \@@_msg_new:nn { bad-value-for-baseline }
8712 {
8713     Bad~value~for~baseline.\\
8714     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
8715     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
8716     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
8717     the~form~'line-i'.\\
8718     A~value~of~1~will~be~used.
8719 }
8720 \@@_msg_new:nn { ragged2e~not~loaded }
8721 {
8722     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
8723     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
8724     \l_keys_key_str'~will~be~used~instead.
8725 }
8726 \@@_msg_new:nn { Invalid-name }
8727 {
8728     Invalid~name.\\
8729     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
8730     \SubMatrix\ of~your~\@@_full_name_env:.\\
8731     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
8732     This~key~will~be~ignored.
8733 }
8734 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
8735 {
8736     Wrong~line.\\
8737     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
8738     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
8739     number~is~not~valid.~It~will~be~ignored.
8740 }
8741 \@@_msg_new:nn { Impossible~delimiter }
8742 {
8743     Impossible~delimiter.\\
8744     It's~impossible~to~draw~the~#1~delimiter~of~your~
8745     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
8746     in~that~column.
8747     \bool_if:NT \l_@@_submatrix_slim_bool
8748     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
8749     This~\token_to_str:N \SubMatrix\ will~be~ignored.
8750 }
8751 \@@_msg_new:nn { width~without~X~columns }
8752 {
8753     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~

```

```

8754     That~key~will~be~ignored.
8755 }

8756 \@@_msg_new:nn { key~multiplicity~with~dotted }
8757 {
8758     Incompatible~keys. \\
8759     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
8760     in~a~'custom-line'.~They~are~incompatible. \\
8761     The~key~'multiplicity'~will~be~discarded.
8762 }

8763 \@@_msg_new:nn { empty~environment }
8764 {
8765     Empty~environment.\\
8766     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
8767 }

8768 \@@_msg_new:nn { No~letter~and~no~command }
8769 {
8770     Erroneous~use.\\
8771     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
8772     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
8773     '~ccommand'~(to~draw~horizontal~rules).\\
8774     However,~you~can~go~on.
8775 }

8776 \@@_msg_new:nn { Forbidden~letter }
8777 {
8778     Forbidden~letter.\\
8779     You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
8780     It~will~be~ignored.
8781 }

8782 \@@_msg_new:nn { Several~letters }
8783 {
8784     Wrong~name.\\
8785     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
8786     have~used~'\l_@@_letter_str').\\
8787     It~will~be~ignored.
8788 }

8789 \@@_msg_new:nn { Delimiter~with~small }
8790 {
8791     Delimiter~forbidden.\\
8792     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
8793     because~the~key~'small'~is~in~force.\\
8794     This~error~is~fatal.
8795 }

8796 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
8797 {
8798     Unknown~cell.\\
8799     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
8800     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
8801     can't~be~executed~because~a~cell~doesn't~exist.\\
8802     This~command~\token_to_str:N \line\ will~be~ignored.
8803 }

8804 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
8805 {
8806     Duplicate~name.\\
8807     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
8808     in~this~\@@_full_name_env:.\
8809     This~key~will~be~ignored.\\
8810     \bool_if:NF \c_@@_messages_for_Overleaf_bool
8811     { For~a~list~of~the~names~already~used,~type~H~<return>. }
8812 }
8813 {

```

```

8814 The~names~already~defined~in~this~\@@_full_name_env:\ are:~
8815 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
8816 }

8817 \@@_msg_new:nn { r-or-l-with-preamble }
8818 {
8819   Erroneous~use.\\
8820   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
8821   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
8822   your~\@@_full_name_env:~
8823   This~key~will~be~ignored.
8824 }

8825 \@@_msg_new:nn { Hdotsfor~in~col~0 }
8826 {
8827   Erroneous~use.\\
8828   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
8829   the~array.~This~error~is~fatal.
8830 }

8831 \@@_msg_new:nn { bad~corner }
8832 {
8833   Bad~corner.\\
8834   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
8835   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
8836   This~specification~of~corner~will~be~ignored.
8837 }

8838 \@@_msg_new:nn { bad~border }
8839 {
8840   Bad~border.\\
8841   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
8842   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
8843   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
8844   also~use~the~key~'tikz'
8845   \IfPackageLoadedTF { tikz }
8846     { }
8847     {~if~you~load~the~LaTeX~package~'tikz'}).\\
8848   This~specification~of~border~will~be~ignored.
8849 }

8850 \@@_msg_new:nn { tikz~key~without~tikz }
8851 {
8852   Tikz~not~loaded.\\
8853   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
8854   \Block'~because~you~have~not~loaded~tikz.~
8855   This~key~will~be~ignored.
8856 }

8857 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
8858 {
8859   Erroneous~use.\\
8860   In~the~\@@_full_name_env:,~you~must~use~the~key~
8861   'last~col'~without~value.\\
8862   However,~you~can~go~on~for~this~time~
8863   (the~value~'\l_keys_value_tl'~will~be~ignored).
8864 }

8865 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
8866 {
8867   Erroneous~use.\\
8868   In~\NiceMatrixoptions,~you~must~use~the~key~
8869   'last~col'~without~value.\\
8870   However,~you~can~go~on~for~this~time~
8871   (the~value~'\l_keys_value_tl'~will~be~ignored).
8872 }

8873 \@@_msg_new:nn { Block~too~large~1 }

```

```

8874 {
8875   Block-too-large.\\
8876   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
8877   too~small~for~that~block. \\
8878 }
8879 \@@_msg_new:nn { Block-too-large-2 }
8880 {
8881   Block-too-large.\\
8882   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8883   \g_@@_static_num_of_col_int\
8884   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
8885   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
8886   (&)~at~the~end~of~the~first~row~of~your~
8887   \@@_full_name_env:.\
8888   This~block~and~maybe~others~will~be~ignored.
8889 }
8890 \@@_msg_new:nn { unknown-column-type }
8891 {
8892   Bad-column-type.\\
8893   The~column~type~'#1'~in~your~\@@_full_name_env:\
8894   is~unknown. \\
8895   This~error~is~fatal.
8896 }
8897 \@@_msg_new:nn { unknown-column-type-S }
8898 {
8899   Bad-column-type.\\
8900   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
8901   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
8902   load~that~package. \\
8903   This~error~is~fatal.
8904 }
8905 \@@_msg_new:nn { tabularnote-forbidden }
8906 {
8907   Forbidden-command.\\
8908   You~can't~use~the~command~\token_to_str:N\tabularnote\
8909   ~here.~This~command~is~available~only~in~
8910   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
8911   the~argument~of~a~command~\token_to_str:N \caption\ included~
8912   in~an~environment~{table}. \\
8913   This~command~will~be~ignored.
8914 }
8915 \@@_msg_new:nn { borders-forbidden }
8916 {
8917   Forbidden-key.\\
8918   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
8919   because~the~option~'rounded-corners'~
8920   is~in~force~with~a~non-zero~value.\\
8921   This~key~will~be~ignored.
8922 }
8923 \@@_msg_new:nn { bottomrule-without-booktabs }
8924 {
8925   booktabs~not~loaded.\\
8926   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
8927   loaded~'booktabs'.\\
8928   This~key~will~be~ignored.
8929 }
8930 \@@_msg_new:nn { enumitem-not-loaded }
8931 {
8932   enumitem~not~loaded.\\
8933   You~can't~use~the~command~\token_to_str:N\tabularnote\
8934   ~because~you~haven't~loaded~'enumitem'.\\

```

```

8935 All~the~commands~\token_to_str:N\tabularnote\ will~be~
8936 ignored~in~the~document.
8937 }
8938 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
8939 {
8940 Tikz~not~loaded.\\
8941 You~have~used~the~key~'tikz'~in~the~definition~of~a~
8942 customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
8943 You~can~go~on~but~you~will~have~another~error~if~you~actually~
8944 use~that~custom~line.
8945 }
8946 \@@_msg_new:nn { tikz~in~borders~without~tikz }
8947 {
8948 Tikz~not~loaded.\\
8949 You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
8950 command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
8951 That~key~will~be~ignored.
8952 }
8953 \@@_msg_new:nn { color~in~custom~line~with~tikz }
8954 {
8955 Erroneous~use.\\
8956 In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
8957 which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
8958 The~key~'color'~will~be~discarded.
8959 }
8960 \@@_msg_new:nn { Wrong~last~row }
8961 {
8962 Wrong~number.\\
8963 You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
8964 \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
8965 If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
8966 last~row.~You~can~avoid~this~problem~by~using~'last~row'~
8967 without~value~(more~compilations~might~be~necessary).
8968 }
8969 \@@_msg_new:nn { Yet~in~env }
8970 {
8971 Nested~environments.\\
8972 Environments~of~nicematrix~can't~be~nested.\\
8973 This~error~is~fatal.
8974 }
8975 \@@_msg_new:nn { Outside~math~mode }
8976 {
8977 Outside~math~mode.\\
8978 The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
8979 (and~not~in~\token_to_str:N \vcenter).\\
8980 This~error~is~fatal.
8981 }
8982 \@@_msg_new:nn { One~letter~allowed }
8983 {
8984 Bad~name.\\
8985 The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
8986 It~will~be~ignored.
8987 }
8988 \@@_msg_new:nn { TabularNote~in~CodeAfter }
8989 {
8990 Environment~{TabularNote}~forbidden.\\
8991 You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
8992 but~*before*~the~\token_to_str:N \CodeAfter.\\
8993 This~environment~{TabularNote}~will~be~ignored.
8994 }

```

```

8995 \@@_msg_new:nn { varwidth~not~loaded }
8996 {
8997   varwidth~not~loaded.\\
8998   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
8999   loaded.\\
9000   Your~column~will~behave~like~'p'.
9001 }
9002 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9003 {
9004   Unknow~key.\\
9005   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9006   \c_@@_available_keys_str
9007 }
9008 {
9009   The~available~keys~are~(in~alphabetic~order):~
9010   color,~
9011   dotted,~
9012   multiplicity,~
9013   sep-color,~
9014   tikz,~and~total-width.
9015 }
9016
9017 \@@_msg_new:nnn { Unknown~key~for~Block }
9018 {
9019   Unknown~key.\\
9020   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9021   \Block.\\ It~will~be~ignored. \\
9022   \c_@@_available_keys_str
9023 }
9024 {
9025   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9026   hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,~
9027   t,~T,~tikz,~transparent~and~vlines.
9028 }
9029 \@@_msg_new:nn { Version~of~siunitx~too~old }
9030 {
9031   siunitx~too~old.\\
9032   You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9033   is~too~old.~You~need~at~least~v3.0.38~and~your~log~file~says:~"siunitx,~
9034   \use:c { ver @ siunitx.sty }". \\
9035   This~error~is~fatal.
9036 }
9037 \@@_msg_new:nnn { Unknown~key~for~Brace }
9038 {
9039   Unknown~key.\\
9040   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9041   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9042   It~will~be~ignored. \\
9043   \c_@@_available_keys_str
9044 }
9045 {
9046   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
9047   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
9048   right-shorten)~and~yshift.
9049 }
9050 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9051 {
9052   Unknown~key.\\
9053   The~key~'\l_keys_key_str'~is~unknown.\\
9054   It~will~be~ignored. \\
9055   \c_@@_available_keys_str
9056 }

```



```

9057 {
9058   The~available~keys~are~(in~alphabetic~order):~
9059   delimiters/color,~
9060   rules~(with~the~subkeys~'color'~and~'width'),~
9061   sub-matrix~(several~subkeys)~
9062   and~xdots~(several~subkeys).~
9063   The~latter~is~for~the~command~\token_to_str:N \line.
9064 }

9065 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9066 {
9067   Unknown~key.\\
9068   The~key~'\l_keys_key_str'~is~unknown.\\
9069   It~will~be~ignored. \\
9070   \c_@@_available_keys_str
9071 }
9072 {
9073   The~available~keys~are~(in~alphabetic~order):~
9074   create-cell-nodes,~
9075   delimiters/color~and~
9076   sub-matrix~(several~subkeys).
9077 }

9078 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9079 {
9080   Unknown~key.\\
9081   The~key~'\l_keys_key_str'~is~unknown.\\
9082   That~key~will~be~ignored. \\
9083   \c_@@_available_keys_str
9084 }
9085 {
9086   The~available~keys~are~(in~alphabetic~order):~
9087   'delimiters/color',~
9088   'extra-height',~
9089   'hlines',~
9090   'hvlines',~
9091   'left-xshift',~
9092   'name',~
9093   'right-xshift',~
9094   'rules'~(with~the~subkeys~'color'~and~'width'),~
9095   'slim',~
9096   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9097   and~'right-xshift').\\
9098 }

9099 \@@_msg_new:nnn { Unknown~key~for~notes }
9100 {
9101   Unknown~key.\\
9102   The~key~'\l_keys_key_str'~is~unknown.\\
9103   That~key~will~be~ignored. \\
9104   \c_@@_available_keys_str
9105 }
9106 {
9107   The~available~keys~are~(in~alphabetic~order):~
9108   bottomrule,~
9109   code-after,~
9110   code-before,~
9111   detect-duplicates,~
9112   enumitem-keys,~
9113   enumitem-keys-para,~
9114   para,~
9115   label-in-list,~
9116   label-in-tabular~and~
9117   style.
9118 }

```

```

9119 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9120 {
9121   Unknown~key.\\
9122   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9123   \token_to_str:N \RowStyle. \\
9124   That~key~will~be~ignored. \\
9125   \c_@@_available_keys_str
9126 }
9127 {
9128   The~available~keys~are~(in~alphabetic~order):~
9129   'bold',~
9130   'cell-space-top-limit',~
9131   'cell-space-bottom-limit',~
9132   'cell-space-limits',~
9133   'color',~
9134   'nb-rows'~and~
9135   'rowcolor'.
9136 }
9137 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9138 {
9139   Unknown~key.\\
9140   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9141   \token_to_str:N \NiceMatrixOptions. \\
9142   That~key~will~be~ignored. \\
9143   \c_@@_available_keys_str
9144 }
9145 {
9146   The~available~keys~are~(in~alphabetic~order):~
9147   allow-duplicate-names,~
9148   caption-above,~
9149   cell-space-bottom-limit,~
9150   cell-space-limits,~
9151   cell-space-top-limit,~
9152   code-for-first-col,~
9153   code-for-first-row,~
9154   code-for-last-col,~
9155   code-for-last-row,~
9156   corners,~
9157   custom-key,~
9158   create-extra-nodes,~
9159   create-medium-nodes,~
9160   create-large-nodes,~
9161   delimiters~(several~subkeys),~
9162   end-of-row,~
9163   first-col,~
9164   first-row,~
9165   hlines,~
9166   hvlines,~
9167   hvlines-except-borders,~
9168   last-col,~
9169   last-row,~
9170   left-margin,~
9171   light-syntax,~
9172   matrix/columns-type,~
9173   notes~(several~subkeys),~
9174   nullify-dots,~
9175   pgf-node-code,~
9176   renew-dots,~
9177   renew-matrix,~
9178   respect-arraystretch,~
9179   right-margin,~
9180   rules~(with~the~subkeys~'color'~and~'width'),~
9181   small,~

```

```

9182     sub-matrix~(several~subkeys),~
9183     vlines,~
9184     xdots~(several~subkeys).
9185 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9186 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9187 {
9188     Unknown~key.\\
9189     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9190     \{NiceArray\}. \\
9191     That~key~will~be~ignored. \\
9192     \c_@@_available_keys_str
9193 }
9194 {
9195     The~available~keys~are~(in~alphabetic~order):~
9196     b,~
9197     baseline,~
9198     c,~
9199     cell-space-bottom-limit,~
9200     cell-space-limits,~
9201     cell-space-top-limit,~
9202     code-after,~
9203     code-for-first-col,~
9204     code-for-first-row,~
9205     code-for-last-col,~
9206     code-for-last-row,~
9207     colortbl-like,~
9208     columns-width,~
9209     corners,~
9210     create-extra-nodes,~
9211     create-medium-nodes,~
9212     create-large-nodes,~
9213     extra-left-margin,~
9214     extra-right-margin,~
9215     first-col,~
9216     first-row,~
9217     hlines,~
9218     hvlines,~
9219     hvlines-except-borders,~
9220     last-col,~
9221     last-row,~
9222     left-margin,~
9223     light-syntax,~
9224     name,~
9225     nullify-dots,~
9226     pgf-node-code,~
9227     renew-dots,~
9228     respect-arraystretch,~
9229     right-margin,~
9230     rules~(with~the~subkeys~'color'~and~'width'),~
9231     small,~
9232     t,~
9233     vlines,~
9234     xdots/color,~
9235     xdots/shorten-start,~
9236     xdots/shorten-end,~
9237     xdots/shorten-and~
9238     xdots/line-style.
9239 }

```

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).

```

9240 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9241 {
9242   Unknown~key.\\
9243   The~key~'\l_keys_key_str'~is~unknown~for~the~
9244   \@@_full_name_env:. \\
9245   That~key~will~be~ignored. \\
9246   \c_@@_available_keys_str
9247 }
9248 {
9249   The~available~keys~are~(in~alphabetic~order):~
9250   b,~
9251   baseline,~
9252   c,~
9253   cell-space-bottom-limit,~
9254   cell-space-limits,~
9255   cell-space-top-limit,~
9256   code-after,~
9257   code-for-first-col,~
9258   code-for-first-row,~
9259   code-for-last-col,~
9260   code-for-last-row,~
9261   colortbl-like,~
9262   columns-type,~
9263   columns-width,~
9264   corners,~
9265   create-extra-nodes,~
9266   create-medium-nodes,~
9267   create-large-nodes,~
9268   extra-left-margin,~
9269   extra-right-margin,~
9270   first-col,~
9271   first-row,~
9272   hlines,~
9273   hvlines,~
9274   hvlines-except-borders,~
9275   l,~
9276   last-col,~
9277   last-row,~
9278   left-margin,~
9279   light-syntax,~
9280   name,~
9281   nullify-dots,~
9282   pgf-node-code,~
9283   r,~
9284   renew-dots,~
9285   respect-arraystretch,~
9286   right-margin,~
9287   rules~(with~the~subkeys~'color'~and~'width'),~
9288   small,~
9289   t,~
9290   vlines,~
9291   xdots/color,~
9292   xdots/shorten-start,~
9293   xdots/shorten-end,~
9294   xdots/shorten~and~
9295   xdots/line-style.
9296 }
9297 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9298 {
9299   Unknown~key.\\
9300   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9301   \{NiceTabular\}. \\
9302   That~key~will~be~ignored. \\

```

```

9303 \c_@@_available_keys_str
9304 }
9305 {
9306   The~available~keys~are~(in~alphabetic~order):~
9307   b,~
9308   baseline,~
9309   c,~
9310   caption,~
9311   cell-space-bottom-limit,~
9312   cell-space-limits,~
9313   cell-space-top-limit,~
9314   code-after,~
9315   code-for-first-col,~
9316   code-for-first-row,~
9317   code-for-last-col,~
9318   code-for-last-row,~
9319   colortbl-like,~
9320   columns-width,~
9321   corners,~
9322   custom-line,~
9323   create-extra-nodes,~
9324   create-medium-nodes,~
9325   create-large-nodes,~
9326   extra-left-margin,~
9327   extra-right-margin,~
9328   first-col,~
9329   first-row,~
9330   hlines,~
9331   hvlines,~
9332   hvlines-except-borders,~
9333   label,~
9334   last-col,~
9335   last-row,~
9336   left-margin,~
9337   light-syntax,~
9338   name,~
9339   notes~(several~subkeys),~
9340   nullify-dots,~
9341   pgf-node-code,~
9342   renew-dots,~
9343   respect-arraystretch,~
9344   right-margin,~
9345   rounded-corners,~
9346   rules~(with~the~subkeys~'color'~and~'width'),~
9347   short-caption,~
9348   t,~
9349   tabularnote,~
9350   vlines,~
9351   xdots/color,~
9352   xdots/shorten-start,~
9353   xdots/shorten-end,~
9354   xdots/shorten-and~
9355   xdots/line-style.
9356 }
9357 \@@_msg_new:nnn { Duplicate-name }
9358 {
9359   Duplicate-name.\\
9360   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9361   the~same~environment~name~twice.~You~can~go~on,~but,~
9362   maybe,~you~will~have~incorrect~results~especially~
9363   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9364   message~again,~use~the~key~'allow-duplicate-names'~in~
9365   '\token_to_str:N \NiceMatrixOptions'.\\

```

```

9366     \c_@@_available_keys_str
9367 }
9368 {
9369     The~names~already~defined~in~this~document~are:~
9370     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9371 }
9372 \@@_msg_new:nn { Option~auto~for~columns~width }
9373 {
9374     Erroneous~use.\\
9375     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
9376     That~key~will~be~ignored.
9377 }

```

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>	69
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	116
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>	146
25	The extra nodes	146
26	The blocks	151
27	How to draw the dotted lines transparently	169
28	Automatic arrays	169
29	The redefinition of the command <code>\dotfill</code>	170
30	The command <code>\diagbox</code>	171

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>	183
35	The command <code>\ShowCellNames</code>	186
36	We process the options at package loading	188
37	About the package underscore	190
38	Error messages of the package	190