

The code of the package `nicematrix`*

F. Pantigny
`fpantigny@wanadoo.fr`

May 15, 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 registred for this package.

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

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

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

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

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

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

We load some packages.

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

*This document corresponds to the version 6.19 of `nicematrix`, at the date of 2023/05/15.

```

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

```

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

```

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

```

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

```

25 \cs_new_protected:Npn \@@_error_or_warning:n
26 { \bool_if:NTF \c_@@_messages_for_Overleaf_bool \@@_warning:n \@@_error:n }

```

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

```

27 \bool_set:Nn \c_@@_messages_for_Overleaf_bool
28 {
29   \str_if_eq_p:Vn \c_sys_jobname_str { _region_ } % for Emacs
30   || \str_if_eq_p:Vn \c_sys_jobname_str { output } % for Overleaf
31 }

```

```

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

```

2 Security test

Within the package `nicematrix`, we will have to test whether a cell of a `{NiceTabular}` is empty. For the cells of the columns of type `p`, `b`, `m`, `X` and `V`, we will test whether the cell is syntactically empty (that is to say that there is only spaces between the ampersands `&`). That test will be done with

the command `\@@_test_if_empty`: by testing if the two first tokens in the cells are (during the TeX process) are `\ignorespaces` and `\unskip`.

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

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

```

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

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

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

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

```

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

```

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

```

3 Technical definitions

```

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

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

```

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

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

```

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

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date March 2023, the current version `revtex4-2` is 4.2e (compatible with `booktabs`).

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`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) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

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

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

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the `code-before`).

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

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

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

```
396 \int_new:N \l_@@_col_max_int
```

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

```
397 \seq_new:N \g_@@_submatrix_seq
```

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble of course and without the potential exterior columns).

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

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

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

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

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

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

```
403 \dim_new:N \l_@@_rounded_corners_dim
```

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

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

```
404 \dim_new:N \l_@@_tab_rounded_corners_dim
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
424 \bool_new:N \l_@@_dotted_bool
```

The following flag will be set to `true` during the composition of a caption specified (by the key `caption`).

```
425 \bool_new:N \l_@@_in_caption_bool
```

Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

- **First row**

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

```
426 \int_new:N \l_@@_first_row_int
427 \int_set:Nn \l_@@_first_row_int 1
```

- **First column**

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

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

- **Last row**

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

```
430 \int_new:N \l_@@_last_row_int
431 \int_set:Nn \l_@@_last_row_int { -2 }
```

If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.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 component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabulernote}`. The first component is the optional argument (between square brackets) of the command `\tabulernote` (if the optional argument is not used, the value will be the special marker `\c_novalue_tl`).
- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

The LaTeX counter `tabulernote` will be used to count the tabular notes during the construction of the array (this counter won't be used during the composition of the notes at the end of the array). You use a LaTeX counter because we will use `\refstepcounter` in order to have the tabular notes referenceable.

```

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

```

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.

```

557   \hbox_set:Nn \l_tmpa_box
558   {

```

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

```

559   \@@_notes_label_in_tabular:n
560   {
561     \seq_use:Nnnn
562     \l_@@_notes_labels_seq { , } { , } { , }
563   }
564 }

```

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

```

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

```

572 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
573 }
574 }

```

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`.

```

575 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
576 {
577   \bool_if:NTF \g_@@_caption_finished_bool
578   {
579     \int_compare:nNnT
580     \c@tabularnote = \g_@@_notes_caption_int
581     { \int_gzero:N \c@tabularnote }

```

Now, we try to detect duplicate notes in the caption.

```

582 \bool_set_false:N \l_tmpa_bool
583 \seq_map_inline:Nn \g_@@_notes_in_caption_seq
584 { \@@_if_eq_two_three:nnn ##1 { #2 } }

```

Be careful! We must put `\bool_if:NF` and not `\bool_if:NT`!

```

585 \bool_if:NF \l_tmpa_bool
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. We have to see whether the text of our tabular note is among the texts of the tabularnotes stored in `\g_@@_notes_in_caption_seq`. For that, we will loop on `\g_@@_notes_in_caption_seq` and we will raise `\l_tmpa_bool` if the text is found.

```

589 \bool_set_false:N \l_tmpa_bool
590 \seq_map_inline:Nn \g_@@_notes_in_caption_seq
591 { \@@_if_eq_two_three:nnn ##1 { #2 } }
592 \bool_if:NTF \l_tmpa_bool
593 {

```

Now, we know that we 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.

```

594 \bool_gset_true:N \g_@@_caption_finished_bool
595 \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
596 \int_gzero:N \c@tabularnote
597 }
598 { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
599 }

```

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!

```

600   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
601   \seq_put_right:Nx \l_@@_notes_labels_seq
602   {
603     \tl_if_novalue:nTF { #1 }
604       { \@@_notes_format:n { \int_use:N \c@tabularnote } }
605       { #1 }
606   }
607   \peek_meaning:NF \tabularnote
608   {
609     \@@_notes_label_in_tabular:n
610     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
611     \seq_clear:N \l_@@_notes_labels_seq
612   }
613 }

614 \cs_new_protected:Npn \@@_if_eq_two_three:nnn #1 #2 #3
615 {
616   \tl_if_eq:nnT { #2 } { #3 }
617   {
618     \bool_set_true:N \l_tmpa_bool
619     \seq_map_break:
620   }
621 }

622 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
623 { \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.

```

624 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
625 {
626   \begin { pgfscope }
627   \pgfset
628   {
629     % outer~sep = \c_zero_dim ,
630     inner~sep = \c_zero_dim ,
631     minimum~size = \c_zero_dim
632   }
633   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
634   \pgfnode
635   { rectangle }
636   { center }
637   {
638     \vbox_to_ht:nn
639     { \dim_abs:n { #5 - #3 } }
640     {
641       \vfill
642       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
643     }
644   }
645   { #1 }
646   { }
647   \end { pgfscope }
648 }

```

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.

```

649 \cs_new_protected:Npn \@@pgf_rect_node:nnn #1 #2 #3
650 {
651   \begin { pgfscope }
652   \pgfset
653   {
654     % outer~sep = \c_zero_dim ,
655     inner~sep = \c_zero_dim ,
656     minimum~size = \c_zero_dim
657   }
658   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
659   \pgfpointdiff { #3 } { #2 }
660   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
661   \pgfnode
662   { rectangle }
663   { center }
664   {
665     \vbox_to_ht:nn
666     { \dim_abs:n \l_tmpb_dim }
667     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
668   }
669   { #1 }
670   { }
671   \end { pgfscope }
672 }

```

7 The options

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

```

673 \tl_new:N \l_@@_caption_tl
674 \tl_new:N \l_@@_short_caption_tl
675 \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).

```

676 \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.

```

677 \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`.

```

678 \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`).

```

679 \dim_new:N \l_@@_cell_space_top_limit_dim
680 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```
681 \dim_new:N \l_@@_xdots_inter_dim
682 \hook_gput_code:nnn { begindocument } { . }
683 { \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).

```
684 \dim_new:N \l_@@_xdots_shorten_start_dim
685 \dim_new:N \l_@@_xdots_shorten_end_dim
686 \hook_gput_code:nnn { begindocument } { . }
687 {
688   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
689   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
690 }
```

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.

```
691 \dim_new:N \l_@@_xdots_radius_dim
692 \hook_gput_code:nnn { begindocument } { . }
693 { \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.

```
694 \tl_new:N \l_@@_xdots_line_style_tl
695 \tl_const:Nn \c_@@_standard_tl { standard }
696 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
697 \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).

```
698 \tl_new:N \l_@@_baseline_tl
699 \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`).

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

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

```
701 \bool_new:N \l_@@_parallelize_diags_bool
702 \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`.

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



```

704 \dim_new:N \l_@@_notes_above_space_dim
705 \hook_gput_code:nnn { begindocument } { . }
706 { \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.).

```

707 \bool_new:N \l_@@_nullify_dots_bool

```

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

```

708 \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).

```

709 \bool_new:N \l_@@_auto_columns_width_bool

```

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

```

710 \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.

```

711 \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”.

```

712 \bool_new:N \l_@@_medium_nodes_bool
713 \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).

```

714 \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`).

```

715 \dim_new:N \l_@@_left_margin_dim
716 \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`.

```

717 \dim_new:N \l_@@_extra_left_margin_dim
718 \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.

```

719 \tl_new:N \l_@@_end_of_row_tl
720 \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 “:”.

```

721 \tl_new:N \l_@@_xdots_color_tl

```

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

```
722 \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.

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

```
724 \keys_define:nn { NiceMatrix / xdots }
725 {
726   line-style .code:n =
727   {
728     \bool_lazy_or:nnTF
729     { \cs_if_exist_p:N \tikzpicture }
730     { \str_if_eq_p:nn { #1 } { standard } }
731     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
732     { \@_error:n { bad-option-for~line-style } }
733   } ,
734   line-style .value_required:n = true ,
735   color .tl_set:N = \l_@@_xdots_color_tl ,
736   color .value_required:n = true ,
737   shorten .code:n =
738     \hook_gput_code:nnn { begindocument } { . }
739     {
740       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
741       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
742     } ,
743   shorten-start .code:n =
744     \hook_gput_code:nnn { begindocument } { . }
745     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
746   shorten-end .code:n =
747     \hook_gput_code:nnn { begindocument } { . }
748     { \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.

```
749   shorten .value_required:n = true ,
750   shorten-start .value_required:n = true ,
751   shorten-end .value_required:n = true ,
752   radius .code:n =
753     \hook_gput_code:nnn { begindocument } { . }
754     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
755   radius .value_required:n = true ,
756   inter .code:n =
757     \hook_gput_code:nnn { begindocument } { . }
758     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
759   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 `_`.

```
760   down .tl_set:N = \l_@@_xdots_down_tl ,
761   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).

```
762   draw-first .code:n = \prg_do_nothing: ,
763   unknown .code:n = \@_error:n { Unknown-key-for~xdots }
764 }
```

```

765 \keys_define:nn { NiceMatrix / rules }
766 {
767   color .tl_set:N = \l_@@_rules_color_tl ,
768   color .value_required:n = true ,
769   width .dim_set:N = \arrayrulewidth ,
770   width .value_required:n = true ,
771   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
772 }

```

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

```

773 \keys_define:nn { NiceMatrix / Global }
774 {
775   custom-line .code:n = \@@_custom_line:n { #1 } ,
776   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
777   rules .value_required:n = true ,
778   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
779   standard-cline .default:n = true ,
780   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
781   cell-space-top-limit .value_required:n = true ,
782   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
783   cell-space-bottom-limit .value_required:n = true ,
784   cell-space-limits .meta:n =
785   {
786     cell-space-top-limit = #1 ,
787     cell-space-bottom-limit = #1 ,
788   } ,
789   cell-space-limits .value_required:n = true ,
790   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
791   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
792   light-syntax .default:n = true ,
793   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
794   end-of-row .value_required:n = true ,
795   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
796   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
797   last-row .int_set:N = \l_@@_last_row_int ,
798   last-row .default:n = -1 ,
799   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
800   code-for-first-col .value_required:n = true ,
801   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
802   code-for-last-col .value_required:n = true ,
803   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
804   code-for-first-row .value_required:n = true ,
805   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
806   code-for-last-row .value_required:n = true ,
807   hlines .clist_set:N = \l_@@_hlines_clist ,
808   vlines .clist_set:N = \l_@@_vlines_clist ,
809   hlines .default:n = all ,
810   vlines .default:n = all ,
811   vl_lines_in_sub_matrix .code:n =
812   {
813     \tl_if_single_token:nTF { #1 }
814     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
815     { \@@_error:n { One-letter-allowed } }
816   } ,
817   vl_lines_in_sub_matrix .value_required:n = true ,
818   hv_lines .code:n =
819   {
820     \bool_set_true:N \l_@@_hv_lines_bool
821     \clist_set:Nn \l_@@_vlines_clist { all }
822     \clist_set:Nn \l_@@_hlines_clist { all }
823   } ,
824   hv_lines_except_borders .code:n =

```

```

825 {
826   \clist_set:Nn \l_@@_vlines_clist { all }
827   \clist_set:Nn \l_@@_hlines_clist { all }
828   \bool_set_true:N \l_@@_hvlines_bool
829   \bool_set_true:N \l_@@_except_borders_bool
830 } ,
831 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.

```

832 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
833 renew-dots .value_forbidden:n = true ,
834 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
835 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
836 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
837 create-extra-nodes .meta:n =
838   { create-medium-nodes , create-large-nodes } ,
839 left-margin .dim_set:N = \l_@@_left_margin_dim ,
840 left-margin .default:n = \arraycolsep ,
841 right-margin .dim_set:N = \l_@@_right_margin_dim ,
842 right-margin .default:n = \arraycolsep ,
843 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
844 margin .default:n = \arraycolsep ,
845 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
846 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
847 extra-margin .meta:n =
848   { extra-left-margin = #1 , extra-right-margin = #1 } ,
849 extra-margin .value_required:n = true ,
850 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
851 respect-arraystretch .default:n = true ,
852 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
853 pgf-node-code .value_required:n = true
854 }

```

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

```

855 \keys_define:nn { NiceMatrix / Env }
856 {
857   corners .clist_set:N = \l_@@_corners_clist ,
858   corners .default:n = { NW , SW , NE , SE } ,
859   code-before .code:n =
860     {
861       \tl_if_empty:nF { #1 }
862       {
863         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
864         \bool_set_true:N \l_@@_code_before_bool
865       }
866     } ,
867   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}`.

```

868 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
869 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
870 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
871 baseline .tl_set:N = \l_@@_baseline_tl ,
872 baseline .value_required:n = true ,
873 columns-width .code:n =
874   \tl_if_eq:nnTF { #1 } { auto }
875     { \bool_set_true:N \l_@@_auto_columns_width_bool }
876     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

```

877 columns-width .value_required:n = true ,
878 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.

```

879 \legacy_if:nF { measuring@ }
880 {
881   \str_set:Nn \l_tmpa_str { #1 }
882   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
883     { \@@_error:nn { Duplicate-name } { #1 } }
884     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
885   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
886 } ,
887 name .value_required:n = true ,
888 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
889 code-after .value_required:n = true ,
890 colortbl-like .code:n =
891   \bool_set_true:N \l_@@_colortbl_like_bool
892   \bool_set_true:N \l_@@_code_before_bool ,
893 colortbl-like .value_forbidden:n = true
894 }
895 \keys_define:nn { NiceMatrix / notes }
896 {
897   para .bool_set:N = \l_@@_notes_para_bool ,
898   para .default:n = true ,
899   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
900   code-before .value_required:n = true ,
901   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
902   code-after .value_required:n = true ,
903   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
904   bottomrule .default:n = true ,
905   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
906   style .value_required:n = true ,
907   label-in-tabular .code:n =
908     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
909   label-in-tabular .value_required:n = true ,
910   label-in-list .code:n =
911     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
912   label-in-list .value_required:n = true ,
913   enumitem-keys .code:n =
914     {
915       \hook_gput_code:nnn { begindocument } { . }
916       {
917         \IfPackageLoadedTF { enumitem }
918           { \setlist* [ tabularnotes ] { #1 } }
919           { }
920       }
921     } ,
922   enumitem-keys .value_required:n = true ,
923   enumitem-keys-para .code:n =
924     {
925       \hook_gput_code:nnn { begindocument } { . }
926       {
927         \IfPackageLoadedTF { enumitem }
928           { \setlist* [ tabularnotes* ] { #1 } }
929           { }
930       }
931     } ,
932   enumitem-keys-para .value_required:n = true ,
933   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
934   detect-duplicates .default:n = true ,
935   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
936 }

```

```

937 \keys_define:nn { NiceMatrix / delimiters }
938 {
939     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
940     max-width .default:n = true ,
941     color .tl_set:N = \l_@@_delimiters_color_tl ,
942     color .value_required:n = true ,
943 }

```

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

```

944 \keys_define:nn { NiceMatrix }
945 {
946     NiceMatrixOptions .inherit:n =
947         { NiceMatrix / Global } ,
948     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
949     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
950     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
951     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
952     SubMatrix / rules .inherit:n = NiceMatrix / rules ,
953     CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
954     CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
955     NiceMatrix .inherit:n =
956         {
957             NiceMatrix / Global ,
958             NiceMatrix / Env ,
959         } ,
960     NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
961     NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
962     NiceTabular .inherit:n =
963         {
964             NiceMatrix / Global ,
965             NiceMatrix / Env
966         } ,
967     NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
968     NiceTabular / rules .inherit:n = NiceMatrix / rules ,
969     NiceTabular / notes .inherit:n = NiceMatrix / notes ,
970     NiceArray .inherit:n =
971         {
972             NiceMatrix / Global ,
973             NiceMatrix / Env ,
974         } ,
975     NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
976     NiceArray / rules .inherit:n = NiceMatrix / rules ,
977     pNiceArray .inherit:n =
978         {
979             NiceMatrix / Global ,
980             NiceMatrix / Env ,
981         } ,
982     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
983     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
984 }

```

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

```

985 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
986 {
987     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
988     delimiters / color .value_required:n = true ,
989     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
990     delimiters / max-width .default:n = true ,
991     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
992     delimiters .value_required:n = true ,

```

```

993 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
994 width .value_required:n = true ,
995 last-col .code:n =
996   \tl_if_empty:nF { #1 }
997   { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
998   \int_zero:N \l_@@_last_col_int ,
999 small .bool_set:N = \l_@@_small_bool ,
1000 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.

```

1001 renew-matrix .code:n = \@@_renew_matrix: ,
1002 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}`.

```

1003 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.

```

1004 columns-width .code:n =
1005   \tl_if_eq:nnTF { #1 } { auto }
1006   { \@@_error:n { Option-auto-for-columns-width } }
1007   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1008 allow-duplicate-names .code:n =
1009   \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1010 allow-duplicate-names .value_forbidden:n = true ,
1011 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1012 notes .value_required:n = true ,
1013 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1014 sub-matrix .value_required:n = true ,
1015 matrix / columns-type .code:n =
1016   \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1017 matrix / columns-type .value_required:n = true ,
1018 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1019 caption-above .default:n = true ,
1020 unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1021 }

```

`\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.

```

1022 \NewDocumentCommand \NiceMatrixOptions { m }
1023 { \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.

```

1024 \keys_define:nn { NiceMatrix / NiceMatrix }
1025 {
1026   last-col .code:n = \tl_if_empty:nTF {#1}
1027   {
1028     \bool_set_true:N \l_@@_last_col_without_value_bool
1029     \int_set:Nn \l_@@_last_col_int { -1 }
1030   }
1031   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1032   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1033   columns-type .value_required:n = true ,

```

```

1034   l .meta:n = { columns-type = l } ,
1035   r .meta:n = { columns-type = r } ,
1036   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1037   delimiters / color .value_required:n = true ,
1038   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1039   delimiters / max-width .default:n = true ,
1040   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1041   delimiters .value_required:n = true ,
1042   small .bool_set:N = \l_@@_small_bool ,
1043   small .value_forbidden:n = true ,
1044   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1045 }

```

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

```

1046 \keys_define:nn { NiceMatrix / NiceArray }
1047 {

```

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.

```

1048   small .bool_set:N = \l_@@_small_bool ,
1049   small .value_forbidden:n = true ,
1050   last-col .code:n = \tl_if_empty:nF { #1 }
1051     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1052     \int_zero:N \l_@@_last_col_int ,
1053   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1054   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1055   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1056 }
1057 \keys_define:nn { NiceMatrix / pNiceArray }
1058 {
1059   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1060   last-col .code:n = \tl_if_empty:nF { #1 }
1061     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1062     \int_zero:N \l_@@_last_col_int ,
1063   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1064   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1065   delimiters / color .value_required:n = true ,
1066   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1067   delimiters / max-width .default:n = true ,
1068   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1069   delimiters .value_required:n = true ,
1070   small .bool_set:N = \l_@@_small_bool ,
1071   small .value_forbidden:n = true ,
1072   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1073   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1074   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1075 }

```

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

```

1076 \keys_define:nn { NiceMatrix / NiceTabular }
1077 {

```

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.

```

1078   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1079     \bool_set_true:N \l_@@_width_used_bool ,
1080   width .value_required:n = true ,
1081   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
1082   rounded-corners .default:n = 4 pt ,

```



```

1083 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1084 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1085 tabularnote .value_required:n = true ,
1086 caption .tl_set:N = \l_@@_caption_tl ,
1087 caption .value_required:n = true ,
1088 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1089 short-caption .value_required:n = true ,
1090 label .tl_set:N = \l_@@_label_tl ,
1091 label .value_required:n = true ,
1092 last-col .code:n = \tl_if_empty:nF {#1}
1093 { \@@_error:n { last-col~non-empty~for~NiceArray } }
1094 \int_zero:N \l_@@_last_col_int ,
1095 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1096 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1097 unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1098 }

```

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}`).

```

1099 \cs_new_protected:Npn \@@_cell_begin:w
1100 {

```

`\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.

```

1101 \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).

```

1102 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

1103 \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.

```

1104 \int_compare:nNnT \c@jCol = 1
1105 { \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).

```

1106 \hbox_set:Nw \l_@@_cell_box
1107 \bool_if:NF \l_@@_NiceTabular_bool
1108 {
1109   \c_math_toggle_token
1110   \bool_if:NT \l_@@_small_bool \scriptstyle
1111 }
1112 \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.

```

1113 \int_compare:nNnTF \c@iRow = 0
1114 {
1115   \int_compare:nNnT \c@jCol > 0
1116   {

```

```

1117         \l_@@_code_for_first_row_tl
1118         \xglobal \colorlet { nicematrix-first-row } { . }
1119     }
1120 }
1121 {
1122     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1123     {
1124         \l_@@_code_for_last_row_tl
1125         \xglobal \colorlet { nicematrix-last-row } { . }
1126     }
1127 }
1128 }

```

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.

```

1129 \cs_new_protected:Npn \@@_begin_of_row:
1130 {
1131     \int_gincr:N \c@iRow
1132     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1133     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1134     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1135     \pgfpicture
1136     \pgfrememberpicturepositiononpagetrue
1137     \pgfcoordinate
1138     { \@@_env: - row - \int_use:N \c@iRow - base }
1139     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1140     \str_if_empty:NF \l_@@_name_str
1141     {
1142         \pgfnodealias
1143         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1144         { \@@_env: - row - \int_use:N \c@iRow - base }
1145     }
1146     \endpgfpicture
1147 }

```

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.

```

1148 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1149 {
1150     \int_compare:nNnTF \c@iRow = 0
1151     {
1152         \dim_gset:Nn \g_@@_dp_row_zero_dim
1153         { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1154         \dim_gset:Nn \g_@@_ht_row_zero_dim
1155         { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1156     }
1157     {
1158         \int_compare:nNnT \c@iRow = 1
1159         {
1160             \dim_gset:Nn \g_@@_ht_row_one_dim
1161             { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1162         }
1163     }
1164 }
1165 \cs_new_protected:Npn \@@_rotate_cell_box:
1166 {
1167     \box_rotate:Nn \l_@@_cell_box { 90 }

```

```

1168 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1169 {
1170     \vbox_set_top:Nn \l_@@_cell_box
1171     {
1172         \vbox_to_zero:n { }
1173         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1174         \box_use:N \l_@@_cell_box
1175     }
1176 }
1177 \bool_gset_false:N \g_@@_rotate_bool
1178 }
1179 \cs_new_protected:Npn \@@_adjust_size_box:
1180 {
1181     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1182     {
1183         \box_set_wd:Nn \l_@@_cell_box
1184         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1185         \dim_gzero:N \g_@@_blocks_wd_dim
1186     }
1187     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1188     {
1189         \box_set_dp:Nn \l_@@_cell_box
1190         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1191         \dim_gzero:N \g_@@_blocks_dp_dim
1192     }
1193     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1194     {
1195         \box_set_ht:Nn \l_@@_cell_box
1196         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1197         \dim_gzero:N \g_@@_blocks_ht_dim
1198     }
1199 }
1200 \cs_new_protected:Npn \@@_cell_end:
1201 {
1202     \@@_math_toggle_token:
1203     \hbox_set_end:
1204     \@@_cell_end_i:
1205 }
1206 \cs_new_protected:Npn \@@_cell_end_i:
1207 {

```

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.

```

1208 \g_@@_cell_after_hook_tl
1209 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1210 \@@_adjust_size_box:
1211 \box_set_ht:Nn \l_@@_cell_box
1212 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1213 \box_set_dp:Nn \l_@@_cell_box
1214 { \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”).

```

1215 \dim_gset:Nn \g_@@_max_cell_width_dim
1216 { \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”.

```

1217 \@@_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.

```

1218 \bool_if:NTF \g_@@_empty_cell_bool
1219 { \box_use_drop:N \l_@@_cell_box }
1220 {
1221   \bool_lazy_or:nnTF
1222     \g_@@_not_empty_cell_bool
1223     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1224     \@@_node_for_cell:
1225     { \box_use_drop:N \l_@@_cell_box }
1226 }
1227 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1228 \bool_gset_false:N \g_@@_empty_cell_bool
1229 \bool_gset_false:N \g_@@_not_empty_cell_bool
1230 }

```

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`).

```

1231 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1232 {
1233   \@@_math_toggle_token:
1234   \hbox_set_end:
1235   \bool_if:NF \g_@@_rotate_bool
1236   {
1237     \hbox_set:Nn \l_@@_cell_box
1238     {
1239       \makebox [ \l_@@_col_width_dim ] [ s ]
1240       { \hbox_unpack_drop:N \l_@@_cell_box }
1241     }
1242   }
1243   \@@_cell_end_i:
1244 }

```

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

```

1245 \pgfset
1246 {
1247   nicematrix / cell-node /.style =
1248   {
1249     inner~sep = \c_zero_dim ,
1250     minimum~width = \c_zero_dim
1251   }
1252 }
1253 \cs_new_protected:Npn \@@_node_for_cell:
1254 {

```

```

1255 \pgfpicture
1256 \pgfsetbaseline \c_zero_dim
1257 \pgfrememberpicturepositiononpagetrue
1258 \pgfset { nicematrix / cell-node }
1259 \pgfnode
1260 { rectangle }
1261 { base }
1262 {

```

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).

```

1263 \set@color
1264 \box_use_drop:N \l_@@_cell_box
1265 }
1266 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1267 { \l_@@_pgf_node_code_tl }
1268 \str_if_empty:NF \l_@@_name_str
1269 {
1270 \pgfnodealias
1271 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1272 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1273 }
1274 \endpgfpicture
1275 }

```

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.

```

1276 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1277 {
1278 \cs_new_protected:Npn \@@_patch_node_for_cell:
1279 {
1280 \hbox_set:Nn \l_@@_cell_box
1281 {
1282 \box_move_up:nn { \box_ht: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 - 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.

```

1287 #1
1288 }
1289 \box_use:N \l_@@_cell_box
1290 \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1291 \hbox_overlap_left:n
1292 {
1293 \pgfsys@markposition
1294 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1295 #1
1296 }
1297 }
1298 }
1299 }

```

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

```

1300 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1301 {
1302 \@@_patch_node_for_cell:n
1303 { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1304 }
1305 { \@@_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.

```
1306 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1307 {
1308   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1309   { \g_@@_#2 _ lines _ tl }
1310   {
1311     \use:c { @@ _ draw _ #2 : nnn }
1312     { \int_use:N \c@iRow }
1313     { \int_use:N \c@jCol }
1314     { \exp_not:n { #3 } }
1315   }
1316 }
1317 \cs_new_protected:Npn \@@_array:n
1318 {
1319   \bool_if:NTF \l_@@_NiceTabular_bool
1320   { \dim_set_eq:NN \col@sep \tabcolsep }
1321   { \dim_set_eq:NN \col@sep \arraycolsep }
1322   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1323   { \cs_set_nopar:Npn \@halignto { } }
1324   { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1325   \@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.

```
1326   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1327 }
1328 \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.

```
1329 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1330 \cs_new_protected:Npn \@@_create_row_node:
1331 {
1332   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1333   {
1334     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1335     \@@_create_row_node_i:
1336   }
1337 }
1338 \cs_new_protected:Npn \@@_create_row_node_i:
1339 {
```

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

```

1340   \hbox
1341   {
1342     \bool_if:NT \l_@@_code_before_bool
1343     {
1344       \vtop
1345       {
1346         \skip_vertical:N 0.5\arrayrulewidth
1347         \pgfsys@markposition
1348         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1349         \skip_vertical:N -0.5\arrayrulewidth
1350       }
1351     }
1352     \pgfpicture
1353     \pgfrememberpicturepositiononpagetrue
1354     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1355     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1356     \str_if_empty:NF \l_@@_name_str
1357     {
1358       \pgfnodealias
1359       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1360       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1361     }
1362     \endpgfpicture
1363   }
1364 }

```

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

```

1365 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1366 \cs_new_protected:Npn \@@_everycr_i:
1367 {
1368   \int_gzero:N \c@jCol
1369   \bool_gset_false:N \g_@@_after_col_zero_bool
1370   \bool_if:NF \g_@@_row_of_col_done_bool
1371   {
1372     \@@_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).

```

1373   \tl_if_empty:NF \l_@@_hlines_clist
1374   {
1375     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1376     {
1377       \exp_args:NNx
1378       \clist_if_in:NnT
1379       \l_@@_hlines_clist
1380       { \int_eval:n { \c@iRow + 1 } }
1381     }
1382   }

```

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.

```

1383   \int_compare:nNnT \c@iRow > { -1 }
1384   {
1385     \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@`.

```

1386       { \hrule height \arrayrulewidth width \c_zero_dim }
1387     }
1388   }
1389 }

```

```

1390     }
1391 }

```

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

```

1392 \cs_set_protected:Npn \@@_newcolumnntype #1
1393 {
1394   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1395   \peek_meaning:NTF [
1396     { \newcol@ #1 }
1397     { \newcol@ #1 [ 0 ] }
1398   }

```

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

```

1399 \cs_set_protected:Npn \@@_renew_dots:
1400 {
1401   \cs_set_eq:NN \ldots \@@_Ldots
1402   \cs_set_eq:NN \cdots \@@_Cdots
1403   \cs_set_eq:NN \vdots \@@_Vdots
1404   \cs_set_eq:NN \ddots \@@_Ddots
1405   \cs_set_eq:NN \iddots \@@_Iddots
1406   \cs_set_eq:NN \dots \@@_Ldots
1407   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1408 }

```

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

```

1409 \cs_new_protected:Npn \@@_colortbl_like:
1410 {
1411   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1412   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1413   \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1414 }

```

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

```

1415 \cs_new_protected:Npn \@@_pre_array_ii:
1416 {

```

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

```

1417   \int_gzero:N \g_@@_total_X_weight_int
1418   \@@_expand_clist:N \l_@@_hlines_clist
1419   \@@_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.

```

1420   \IfPackageLoadedTF { booktabs }
1421     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1422     { }
1423   \box_clear_new:N \l_@@_cell_box
1424   \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}`).

```

1425 \bool_if:NT \l_@@_small_bool
1426 {
1427     \cs_set_nopar:Npn \arraystretch { 0.47 }
1428     \dim_set:Nn \arraycolsep { 1.45 pt }
1429 }

1430 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1431 {
1432     \tl_put_right:Nn \@@_begin_of_row:
1433     {
1434         \pgfsys@markposition
1435         { \@@_env: - row - \int_use:N \c@iRow - base }
1436     }
1437 }

```

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`.

```

1438 \cs_set_nopar:Npn \ialign
1439 {
1440     \IfPackageLoadedTF { colortbl }
1441     {
1442         \CT@everycr
1443         {
1444             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1445             \@@_everycr:
1446         }
1447     }
1448     { \everycr { \@@_everycr: } }
1449     \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`.

```

1450 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1451 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1452 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1453 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1454 \dim_gzero_new:N \g_@@_ht_row_one_dim
1455 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1456 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1457 \dim_gzero_new:N \g_@@_ht_last_row_dim
1458 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1459 \dim_gzero_new:N \g_@@_dp_last_row_dim
1460 \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`.

```

1461 \cs_set_eq:NN \ialign \@@_old_ialign:
1462 \halign
1463 }

```

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).

```

1464 \cs_set_eq:NN \@@_old_ldots \ldots
1465 \cs_set_eq:NN \@@_old_cdots \cdots
1466 \cs_set_eq:NN \@@_old_vdots \vdots
1467 \cs_set_eq:NN \@@_old_ddots \ddots
1468 \cs_set_eq:NN \@@_old_iddots \iddots
1469 \bool_if:NTF \l_@@_standard_cline_bool
1470 { \cs_set_eq:NN \cline \@@_standard_cline }
1471 { \cs_set_eq:NN \cline \@@_cline }
1472 \cs_set_eq:NN \Ldots \@@_Ldots
1473 \cs_set_eq:NN \Cdots \@@_Cdots
1474 \cs_set_eq:NN \Vdots \@@_Vdots
1475 \cs_set_eq:NN \Ddots \@@_Ddots
1476 \cs_set_eq:NN \Iddots \@@_Iddots
1477 \cs_set_eq:NN \Hline \@@_Hline:
1478 \cs_set_eq:NN \Hspace \@@_Hspace:
1479 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1480 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1481 \cs_set_eq:NN \Block \@@_Block:
1482 \cs_set_eq:NN \rotate \@@_rotate:
1483 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1484 \cs_set_eq:NN \dotfill \@@_dotfill:
1485 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1486 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1487 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1488 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1489 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1490 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1491 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1492 \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.

```

1493 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1494 \hook_gput_code:nnn { env / tabular / begin } { . }
1495 { \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!).

```

1496 \tl_if_exist:NT \l_@@_note_in_caption_tl
1497 {
1498   \tl_if_empty:NF \l_@@_note_in_caption_tl
1499   {
1500     \int_gset_eq:NN \g_@@_notes_caption_int
1501     { \l_@@_note_in_caption_tl }
1502     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1503   }
1504 }

```

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).

```

1505 \seq_gclear:N \g_@@_multicolumn_cells_seq
1506 \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).

```
1507 \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).

```
1508 \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.

```
1509 \int_gzero_new:N \g_@@_col_total_int
```

```
1510 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

```
1511 \@@_renew_NC@rewrite@S:
```

```
1512 \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.

```
1513 \tl_gclear_new:N \g_@@_Cdots_lines_tl
```

```
1514 \tl_gclear_new:N \g_@@_Ldots_lines_tl
```

```
1515 \tl_gclear_new:N \g_@@_Vdots_lines_tl
```

```
1516 \tl_gclear_new:N \g_@@_Ddots_lines_tl
```

```
1517 \tl_gclear_new:N \g_@@_Iddots_lines_tl
```

```
1518 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl
```

```
1519 \tl_gclear:N \g_nicematrix_code_before_tl
```

```
1520 \tl_gclear:N \g_@@_pre_code_before_tl
```

```
1521 }
```

This is the end of \@@_pre_array_ii:.

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

```
1522 \cs_new_protected:Npn \@@_pre_array:
```

```
1523 {
```

```
1524 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
```

```
1525 \int_gzero_new:N \c@iRow
```

```
1526 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
```

```
1527 \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).

```
1528 \int_compare:nNnT \l_@@_last_row_int = { -1 }
```

```
1529 {
```

```
1530 \bool_set_true:N \l_@@_last_row_without_value_bool
```

```
1531 \bool_if:NT \g_@@_aux_found_bool
```

```
1532 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
```

```
1533 }
```

```
1534 \int_compare:nNnT \l_@@_last_col_int = { -1 }
```

```
1535 {
```

```
1536 \bool_if:NT \g_@@_aux_found_bool
```

```
1537 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
```

```
1538 }
```

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”.

```

1539   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1540   {
1541     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1542     {
1543       \dim_gset:Nn \g_@@_ht_last_row_dim
1544       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1545       \dim_gset:Nn \g_@@_dp_last_row_dim
1546       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1547     }
1548   }

1549   \seq_gclear:N \g_@@_cols_vlism_seq
1550   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1551   \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.

```

1552   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1553   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1554   \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.

```

1555   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1556   \@@_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.

```

1557   \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]`).

```

1558   \dim_zero_new:N \l_@@_left_delim_dim
1559   \dim_zero_new:N \l_@@_right_delim_dim
1560   \bool_if:NTF \g_@@_NiceArray_bool
1561   {
1562     \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1563     \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1564   }
1565   {

```

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

```

1566   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1567   \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1568   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1569   \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1570   }

```

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).

```

1571 \hbox_set:Nw \l_@@_the_array_box
1572 \skip_horizontal:N \l_@@_left_margin_dim
1573 \skip_horizontal:N \l_@@_extra_left_margin_dim
1574 \c_math_toggle_token
1575 \bool_if:NTF \l_@@_light_syntax_bool
1576 { \use:c { @@-light-syntax } }
1577 { \use:c { @@-normal-syntax } }
1578 }

```

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

```

1579 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1580 {
1581   \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1582   \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...

```

1583 \@@_pre_array:
1584 }

```

9 The `\CodeBefore`

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

```

1585 \cs_new_protected:Npn \@@_pre_code_before:
1586 {

```

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).

```

1587 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1588 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1589 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1590 \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.

```

1591 \pgfsys@markposition { \@@_env: - position }
1592 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1593 \pgfpicture
1594 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1595 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1596 {
1597   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1598   \pgfcoordinate { \@@_env: - row - ##1 }
1599   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1600 }

```

Now, the recreation of the col nodes.

```

1601 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1602 {
1603   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1604   \pgfcoordinate { \@@_env: - col - ##1 }
1605   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1606 }

```

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

```

1607 \@@_create_diag_nodes:

```

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

```

1608 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1609 \endpgfpicture

```

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

```

1610 \@@_create_blocks_nodes:
1611 \IfPackageLoadedTF { tikz }
1612 {
1613   \tikzset
1614   {
1615     every~picture / .style =
1616     { overlay , name~prefix = \@@_env: - }
1617   }
1618 }
1619 { }
1620 \cs_set_eq:NN \cellcolor \@@_cellcolor
1621 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1622 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1623 \cs_set_eq:NN \rowcolor \@@_rowcolor
1624 \cs_set_eq:NN \rowcolors \@@_rowcolors
1625 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1626 \cs_set_eq:NN \arraycolor \@@_arraycolor
1627 \cs_set_eq:NN \columncolor \@@_columncolor
1628 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1629 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1630 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1631 }

1632 \cs_new_protected:Npn \@@_exec_code_before:
1633 {
1634   \seq_gclear_new:N \g_@@_colors_seq
1635   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1636   \group_begin:

```

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

```

1637 \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`).

```

1638 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1639 {
1640   \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1641   \@@_rescan_for_spanish:N \l_@@_code_before_tl
1642 }

```

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`.

```
1643 \exp_last_unbraced:NV \@@_CodeBefore_keys:
1644 \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.

```
1645 \@@_actually_color:
1646 \l_@@_code_before_tl
1647 \q_stop
1648 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1649 \group_end:
1650 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1651 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1652 }
```

```
1653 \keys_define:nn { NiceMatrix / CodeBefore }
1654 {
1655   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1656   create-cell-nodes .default:n = true ,
1657   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1658   sub-matrix .value_required:n = true ,
1659   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1660   delimiters / color .value_required:n = true ,
1661   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1662 }

1663 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1664 {
1665   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1666   \@@_CodeBefore:w
1667 }
```

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.

```
1668 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1669 {
1670   \bool_if:NT \g_@@_aux_found_bool
1671   {
1672     \@@_pre_code_before:
1673     #1
1674   }
1675 }
```

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.

```
1676 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1677 {
1678   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1679   {
1680     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1681     \pgfcoordinate { \@@_env: - row - ##1 - base }
1682     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1683     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
```

```

1684 {
1685   \cs_if_exist:cT
1686   { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1687   {
1688     \pgfsys@getposition
1689     { \@@_env: - ##1 - #####1 - NW }
1690     \@@_node_position:
1691     \pgfsys@getposition
1692     { \@@_env: - ##1 - #####1 - SE }
1693     \@@_node_position_i:
1694     \@@_pgf_rect_node:nnn
1695     { \@@_env: - ##1 - #####1 }
1696     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1697     { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1698   }
1699 }
1700 }
1701 \int_step_inline:nn \c@iRow
1702 {
1703   \pgfnodealias
1704   { \@@_env: - ##1 - last }
1705   { \@@_env: - ##1 - \int_use:N \c@jCol }
1706 }
1707 \int_step_inline:nn \c@jCol
1708 {
1709   \pgfnodealias
1710   { \@@_env: - last - ##1 }
1711   { \@@_env: - \int_use:N \c@iRow - ##1 }
1712 }
1713 \@@_create_extra_nodes:
1714 }

1715 \cs_new_protected:Npn \@@_create_blocks_nodes:
1716 {
1717   \pgfpicture
1718   \pgf@relevantforpicturesizefalse
1719   \pgfrememberpicturepositiononpagetrue
1720   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1721   { \@@_create_one_block_node:nnnnn ##1 }
1722   \endpgfpicture
1723 }

```

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

```

1724 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1725 {
1726   \tl_if_empty:nF { #5 }
1727   {
1728     \@@_qpoint:n { col - #2 }
1729     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1730     \@@_qpoint:n { #1 }
1731     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1732     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1733     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1734     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1735     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1736     \@@_pgf_rect_node:nnnnn
1737     { \@@_env: - #5 }
1738     { \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).


```

1739         { \dim_use:N \l_tmpb_dim }
1740         { \dim_use:N \l_@@_tmpc_dim }
1741         { \dim_use:N \l_@@_tmpd_dim }
1742     }
1743 }

1744 \cs_new_protected:Npn \@@_patch_for_revtext:
1745 {
1746     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1747     \cs_set_eq:NN \insert@column \insert@column@array
1748     \cs_set_eq:NN \@classx \@classx@array
1749     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1750     \cs_set_eq:NN \@arraycr \@arraycr@array
1751     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1752     \cs_set_eq:NN \array \array@array
1753     \cs_set_eq:NN \@array \@array@array
1754     \cs_set_eq:NN \@tabular \@tabular@array
1755     \cs_set_eq:NN \@mkpream \@mkpream@array
1756     \cs_set_eq:NN \endarray \endarray@array
1757     \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1758     \cs_set:Npn \endtabular { \endarray $\egroup} % $
1759 }

```

10 The environment `{NiceArrayWithDelims}`

```

1760 \NewDocumentEnvironment { NiceArrayWithDelims }
1761 { m m O { } m ! O { } t \CodeBefore }
1762 {
1763     \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1764     \@@_provide_pgfsyspdfmark:
1765     \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.

```

1766     \bgroup

1767     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1768     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1769     \tl_gset:Nn \g_@@_preamble_tl { #4 }

1770     \int_gzero:N \g_@@_block_box_int
1771     \dim_zero:N \g_@@_width_last_col_dim
1772     \dim_zero:N \g_@@_width_first_col_dim
1773     \bool_gset_false:N \g_@@_row_of_col_done_bool
1774     \str_if_empty:NT \g_@@_name_env_str
1775     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1776     \bool_if:NTF \l_@@_NiceTabular_bool
1777     \mode_leave_vertical:
1778     \@@_test_if_math_mode:
1779     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1780     \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.

```
1781 \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.

```
1782 \cs_if_exist:NT \tikz@library@external@loaded
1783 {
1784   \tikzexternaldisable
1785   \cs_if_exist:NT \ifstandalone
1786   { \tikzset { external / optimize = false } }
1787 }
```

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

```
1788 \int_gincr:N \g_@@_env_int
1789 \bool_if:NF \l_@@_block_auto_columns_width_bool
1790 { \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`).

```
1791 \seq_gclear:N \g_@@_blocks_seq
1792 \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`.

```
1793 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1794 \seq_gclear:N \g_@@_pos_of_xdots_seq
1795 \tl_gclear_new:N \g_@@_code_before_tl
1796 \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.

```
1797 \bool_gset_false:N \g_@@_aux_found_bool
1798 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1799 {
1800   \bool_gset_true:N \g_@@_aux_found_bool
1801   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1802 }
```

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.

```
1803 \tl_gclear:N \g_@@_aux_tl
1804 \tl_if_empty:NF \g_@@_code_before_tl
1805 {
1806   \bool_set_true:N \l_@@_code_before_bool
1807   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1808 }
1809 \tl_if_empty:NF \g_@@_pre_code_before_tl
1810 { \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`.

```
1811 \bool_if:NTF \g_@@_NiceArray_bool
1812 { \keys_set:nn { NiceMatrix / NiceArray } }
1813 { \keys_set:nn { NiceMatrix / pNiceArray } }
1814 { #3 , #5 }

1815 \@@_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:.`

```
1816   \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1817 }
```

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

```
1818 {
1819   \bool_if:NTF \l_@@_light_syntax_bool
1820   { \use:c { end @@-light-syntax } }
1821   { \use:c { end @@-normal-syntax } }
1822   \c_math_toggle_token
1823   \skip_horizontal:N \l_@@_right_margin_dim
1824   \skip_horizontal:N \l_@@_extra_right_margin_dim
1825   \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.

```
1826   \bool_if:NT \l_@@_width_used_bool
1827   {
1828     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1829     { \@@_error_or_warning:n { width-without-X-columns } }
1830   }
```

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 .

```
1831   \int_compare:nNnT \g_@@_total_X_weight_int > 0
1832   {
1833     \tl_gput_right:Nx \g_@@_aux_tl
1834     {
1835       \bool_set_true:N \l_@@_X_columns_aux_bool
1836       \dim_set:Nn \l_@@_X_columns_dim
1837       {
1838         \dim_compare:nNnTF
1839         {
1840           \dim_abs:n
1841             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1842         }
1843         <
1844         { 0.001 pt }
1845         { \dim_use:N \l_@@_X_columns_dim }
1846         {
1847           \dim_eval:n
1848           {
1849             ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1850             / \int_use:N \g_@@_total_X_weight_int
1851             + \l_@@_X_columns_dim
1852           }
1853         }
1854       }
1855     }
1856   }
```

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).

```
1857   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1858   {
1859     \bool_if:NF \l_@@_last_row_without_value_bool
1860     {
1861       \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1862       {
```

```

1863         \@@_error:n { Wrong~last~row }
1864         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1865     }
1866 }
1867 }

```

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

```

1868     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1869     \bool_if:nTF \g_@@_last_col_found_bool
1870     { \int_gdecr:N \c@jCol }
1871     {
1872         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1873         { \@@_error:n { last~col~not~used } }
1874     }

```

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

```

1875     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1876     \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).

```

1877     \int_compare:nNnT \l_@@_first_col_int = 0
1878     {
1879         \skip_horizontal:N \col@sep
1880         \skip_horizontal:N \g_@@_width_first_col_dim
1881     }

```

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.

```

1882     \bool_if:nTF \g_@@_NiceArray_bool
1883     {
1884         \str_case:VnF \l_@@_baseline_tl
1885         {
1886             b \@@_use_arraybox_with_notes_b:
1887             c \@@_use_arraybox_with_notes_c:
1888         }
1889         \@@_use_arraybox_with_notes:
1890     }

```

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).

```

1891     {
1892         \int_compare:nNnTF \l_@@_first_row_int = 0
1893         {
1894             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1895             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1896         }
1897         { \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

```

1898     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1899     {
1900         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1901         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1902     }
1903     { \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).

```

1904     \hbox_set:Nn \l_tmpa_box
1905     {
1906         \c_math_toggle_token
1907         \@@_color:V \l_@@_delimiters_color_tl
1908         \exp_after:wN \left \g_@@_left_delim_tl
1909         \vcenter
1910         {

```

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.

```

1911         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1912         \hbox
1913         {
1914             \bool_if:NTF \l_@@_NiceTabular_bool
1915             { \skip_horizontal:N -\tabcolsep }
1916             { \skip_horizontal:N -\arraycolsep }
1917             \@@_use_arraybox_with_notes_c:
1918             \bool_if:NTF \l_@@_NiceTabular_bool
1919             { \skip_horizontal:N -\tabcolsep }
1920             { \skip_horizontal:N -\arraycolsep }
1921         }

```

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

```

1922         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1923     }

```

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.

```

1924         \@@_color:V \l_@@_delimiters_color_tl
1925         \exp_after:wN \right \g_@@_right_delim_tl
1926         \c_math_toggle_token
1927     }

```

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.

```

1928     \bool_if:NTF \l_@@_delimiters_max_width_bool
1929     {
1930         \@@_put_box_in_flow_bis:nn
1931         \g_@@_left_delim_tl \g_@@_right_delim_tl
1932     }
1933     \@@_put_box_in_flow:
1934 }

```

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).

```

1935     \bool_if:NT \g_@@_last_col_found_bool
1936     {
1937         \skip_horizontal:N \g_@@_width_last_col_dim
1938         \skip_horizontal:N \col@sep
1939     }
1940     \bool_if:NF \l_@@_Matrix_bool
1941     {
1942         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1943         { \@@_warning_gredirect_none:n { columns-not-used } }
1944     }
1945     \@@_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.

```

1946     \egroup

```

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

```

1947     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1948     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1949     \iow_now:Nx \@mainaux

```

```

1950     {
1951       \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1952       { \exp_not:V \g_@@_aux_tl }
1953     }
1954     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1955     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1956   }

```

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.

```

1957 \cs_new_protected:Npn \@@_transform_preamble:
1958 {

```

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.

```

1959   \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).

```

1960   \bool_if:NF \l_@@_Matrix_bool
1961   {
1962     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1963     \@@_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).

```

1964     \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).

```

1965     \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1966     \@tempswatruue

```

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

```

1967     \@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`.

```

1968      \int_gzero:N \c@jCol
1969      \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1970      \bool_gset_false:N \g_tmpb_bool
1971      \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1972      {
1973          \tl_gset:Nn \g_@@_preamble_tl
1974              { ! { \skip_horizontal:N \arrayrulewidth } }
1975      }
1976      {
1977          \clist_if_in:NnT \l_@@_vlines_clist 1
1978          {
1979              \tl_gset:Nn \g_@@_preamble_tl
1980                  { ! { \skip_horizontal:N \arrayrulewidth } }
1981          }
1982      }

```

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`).

```

1983      \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1984      \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

1985      \int_zero:N \l_tmpa_int

```

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

```

1986      \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1987      \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1988      }

```

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

```

1989      \bool_if:NT \l_@@_colortbl_like_bool
1990      {
1991          \regex_replace_all:NnN
1992              \c_@@_columncolor_regex
1993              { \c { @@_columncolor_preamble } }
1994              \g_@@_preamble_tl
1995      }

```

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

```

1996      \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}`.

```

1997      \bool_lazy_or:nnT
1998          { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1999          { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2000      { \bool_gset_false:N \g_@@_NiceArray_bool }

```

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

```

2001      \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).

```

2002 \int_compare:nNnTF \l_@@_first_col_int = 0
2003 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
2004 {
2005   \bool_lazy_all:nT
2006   {
2007     \g_@@_NiceArray_bool
2008     { \bool_not_p:n \l_@@_NiceTabular_bool }
2009     { \tl_if_empty_p:N \l_@@_vlines_clist }
2010     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2011   }
2012   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
2013 }
2014 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2015 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
2016 {
2017   \bool_lazy_all:nT
2018   {
2019     \g_@@_NiceArray_bool
2020     { \bool_not_p:n \l_@@_NiceTabular_bool }
2021     { \tl_if_empty_p:N \l_@@_vlines_clist }
2022     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2023   }
2024   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
2025 }

```

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`).

```

2026 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2027 {
2028   \tl_gput_right:Nn \g_@@_preamble_tl
2029   { > { \@@_error_too_much_cols: } 1 }
2030 }
2031 }

```

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

```

2032 \cs_new_protected:Npn \@@_patch_preamble:n #1
2033 {
2034   \str_case:nnF { #1 }
2035   {
2036     c { \@@_patch_preamble_i:n #1 }
2037     l { \@@_patch_preamble_i:n #1 }
2038     r { \@@_patch_preamble_i:n #1 }
2039     > { \@@_patch_preamble_xiv:n }
2040     ! { \@@_patch_preamble_ii:nn #1 }
2041     @ { \@@_patch_preamble_ii:nn #1 }
2042     | { \@@_patch_preamble_iii:nn #1 }
2043     p { \@@_patch_preamble_iv:n #1 }
2044     b { \@@_patch_preamble_iv:n #1 }
2045     m { \@@_patch_preamble_iv:n #1 }
2046     \@@_V: { \@@_patch_preamble_v:n }
2047     V { \@@_patch_preamble_v:n }
2048     \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
2049     \@@_W: { \@@_patch_preamble_vi:nnnn { \@@_special_W: } #1 }
2050     \@@_S: { \@@_patch_preamble_vii:n }
2051     ( { \@@_patch_preamble_viii:nn #1 }
2052     [ { \@@_patch_preamble_viii:nn #1 }
2053     \{ { \@@_patch_preamble_viii:nn #1 }
2054     \left { \@@_patch_preamble_viii:nn }
2055     ) { \@@_patch_preamble_ix:nn #1 }
2056     ] { \@@_patch_preamble_ix:nn #1 }

```



```

2057     \}      { \@@_patch_preamble_ix:nn #1 }
2058     \right { \@@_patch_preamble_ix:nn }
2059     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}`.

```

2060     \@@_X { \@@_patch_preamble_x:n }
2061     \q_stop { }
2062 }
2063 {
2064     \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2065     {
2066         \seq_gput_right:Nx \g_@@_cols_vlism_seq
2067         { \int_eval:n { \c@jCol + 1 } }
2068         \tl_gput_right:Nx \g_@@_preamble_tl
2069         { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2070         \@@_patch_preamble:n
2071     }

```

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).

```

2072     {
2073         \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2074         {
2075             \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2076             \@@_patch_preamble:n
2077         }
2078         {
2079             \tl_if_eq:nnT { #1 } { S }
2080             { \@@_fatal:n { unknown~column~type~S } }
2081             { \@@_fatal:nn { unknown~column~type } { #1 } }
2082         }
2083     }
2084 }
2085 }

```

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`

```

2086 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2087 {
2088     \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2089     \tl_gclear:N \g_@@_pre_cell_tl
2090     \tl_gput_right:Nn \g_@@_preamble_tl
2091     {
2092         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2093         #1
2094         < \@@_cell_end:
2095     }

```

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

```

2096     \int_gincr:N \c@jCol
2097     \@@_patch_preamble_xi:n
2098 }

```

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

```

2099 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2100 {

```

```

2101 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2102 \@@_patch_preamble:n
2103 }

```

For |

```

2104 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2105 {
2106 \l_tmpa_int is the number of successive occurrences of |
2107 \int_incr:N \l_tmpa_int
2108 \@@_patch_preamble_iii_i:n
2109 }

```

```

2109 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2110 {
2111 \str_if_eq:nnTF { #1 } |
2112 { \@@_patch_preamble_iii:n | }
2113 {
2114 \dim_set:Nn \l_tmpa_dim
2115 {
2116 \arrayrulewidth * \l_tmpa_int
2117 + \doublerulesep * ( \l_tmpa_int - 1 )
2118 }
2119 \tl_gput_right:Nx \g_@@_preamble_tl
2120 {

```

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

```

2121 \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_tmpa_dim } } }
2122 }
2123 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2124 {
2125 \@@_vline:n
2126 {
2127 position = \int_eval:n { \c@jCol + 1 } ,
2128 multiplicity = \int_use:N \l_tmpa_int ,
2129 total-width = \dim_use:N \l_tmpa_dim % added 2022-08-06
2130 }

```

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.

```

2131 }
2132 \int_zero:N \l_tmpa_int
2133 \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2134 \@@_patch_preamble:n #1
2135 }
2136 }
2137 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2138 {
2139 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2140 \@@_patch_preamble:n
2141 }
2142 \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.

```

2143 \keys_define:nn { WithArrows / p-column }
2144 {
2145 r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2146 r .value_forbidden:n = true ,
2147 c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2148 c .value_forbidden:n = true ,
2149 l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2150 l .value_forbidden:n = true ,

```

```

2151 R .code:n =
2152     \IfPackageLoadedTF { ragged2e }
2153     { \str_set:Nn \l_@@_hpos_col_str { R } }
2154     {
2155         \@@_error_or_warning:n { ragged2e~not~loaded }
2156         \str_set:Nn \l_@@_hpos_col_str { r }
2157     } ,
2158 R .value_forbidden:n = true ,
2159 L .code:n =
2160     \IfPackageLoadedTF { ragged2e }
2161     { \str_set:Nn \l_@@_hpos_col_str { L } }
2162     {
2163         \@@_error_or_warning:n { ragged2e~not~loaded }
2164         \str_set:Nn \l_@@_hpos_col_str { l }
2165     } ,
2166 L .value_forbidden:n = true ,
2167 C .code:n =
2168     \IfPackageLoadedTF { ragged2e }
2169     { \str_set:Nn \l_@@_hpos_col_str { C } }
2170     {
2171         \@@_error_or_warning:n { ragged2e~not~loaded }
2172         \str_set:Nn \l_@@_hpos_col_str { c }
2173     } ,
2174 C .value_forbidden:n = true ,
2175 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2176 S .value_forbidden:n = true ,
2177 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2178 p .value_forbidden:n = true ,
2179 t .meta:n = p ,
2180 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2181 m .value_forbidden:n = true ,
2182 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2183 b .value_forbidden:n = true ,
2184 }

```

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

```

2185 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2186 {
2187     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2188     \@@_patch_preamble_iv_i:n
2189 }

2190 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2191 {
2192     \str_if_eq:nnTF { #1 } { [ ]
2193         { \@@_patch_preamble_iv_ii:w [ ]
2194             { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2195         }
2196     \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2197     { \@@_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.

```

2198 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2199 {

```

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).

```

2200     \str_set:Nn \l_@@_hpos_col_str { j }
2201     \tl_set:Nn \l_tmpa_tl { #1 }
2202     \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2203     \@@_keys_p_column:V \l_tmpa_tl

```

```

2204 \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2205 }
2206 \cs_new_protected:Npn \@@_keys_p_column:n #1
2207 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2208 \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`.

```

2209 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2210 {
2211   \use:x
2212   {
2213     \@@_patch_preamble_iv_v:nnnnnnnn
2214     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2215     { \dim_eval:n { #1 } }
2216     {

```

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.

```

2217       \str_if_eq:VnTF \l_@@_hpos_col_str j
2218       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
2219       {
2220         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2221         { \str_lowercase:V \l_@@_hpos_col_str }
2222       }
2223     \str_case:Vn \l_@@_hpos_col_str
2224     {
2225       c { \exp_not:N \centering }
2226       l { \exp_not:N \raggedright }
2227       r { \exp_not:N \raggedleft }
2228       C { \exp_not:N \Centering }
2229       L { \exp_not:N \RaggedRight }
2230       R { \exp_not:N \RaggedLeft }
2231     }
2232   }
2233   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2234   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2235   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2236   { #2 }
2237   {
2238     \str_case:VnF \l_@@_hpos_col_str
2239     {
2240       { j } { c }
2241       { si } { c }
2242     }

```

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

```

2243       { \str_lowercase:V \l_@@_hpos_col_str }
2244     }
2245   }

```

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

```

2246   \int_gincr:N \c@jCol
2247   \@@_patch_preamble_xi:n
2248 }

```

#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*.

```

2249 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2250 {
2251   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2252     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2253     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2254   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2255   \tl_gclear:N \g_@@_pre_cell_tl
2256   \tl_gput_right:Nn \g_@@_preamble_tl
2257     {
2258       > {

```

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.

```

2259       \dim_set:Nn \l_@@_col_width_dim { #2 }
2260       \@@_cell_begin:w
2261       \begin { #7 } [ #1 ] { #2 }

```

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

```

2262       \everypar
2263       {
2264         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2265         \everypar { }
2266       }

```

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

```

2267       #3

```

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

```

2268       \g_@@_row_style_tl
2269       \arraybackslash
2270       #5
2271     }
2272     #8
2273     < {
2274       #6

```

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

```

2275       \@finalstrut \@arstrutbox
2276       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
2277       \end { #7 }

```

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

```

2278       #4
2279       \@@_cell_end:
2280     }
2281   }
2282 }

```

```

2283 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2284 {
2285   \peek_meaning:NT \unskip
2286   {
2287     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2288     {
2289       \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.

```

2290         \skip_horizontal:N \l_@@_col_width_dim
2291     }
2292 }
2293 #1
2294 }
2295 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2296 {
2297     \peek_meaning:NT \__siunitx_table_skip:n
2298     {
2299         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2300         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2301     }
2302     #1
2303 }

```

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

```

2304 \cs_new_protected:Npn \@@_center_cell_box:
2305 {

```

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

```

2306     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2307     {
2308         \int_compare:nNnT
2309         { \box_ht:N \l_@@_cell_box }
2310         >

```

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).

```

2311         { \box_ht:N \strutbox }
2312     {
2313         \hbox_set:Nn \l_@@_cell_box
2314         {
2315             \box_move_down:nn
2316             {
2317                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2318                 + \baselineskip ) / 2
2319             }
2320             { \box_use:N \l_@@_cell_box }
2321         }
2322     }
2323 }
2324 }

```

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

```

2325 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2326 {
2327     \str_if_eq:nnTF { #1 } { [ ] }
2328     { \@@_patch_preamble_v_i:w [ ] }
2329     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2330 }
2331 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2332 { \@@_patch_preamble_v_ii:nn { #1 } }
2333 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2334 {

```

```

2335 \str_set:Nn \l_@@_vpos_col_str { p }
2336 \str_set:Nn \l_@@_hpos_col_str { j }
2337 \tl_set:Nn \l_tmpa_tl { #1 }
2338 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2339 \@@_keys_p_column:V \l_tmpa_tl
2340 \IfPackageLoadedTF { varwidth }
2341 { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2342 {
2343   \@@_error_or_warning:n { varwidth~not~loaded }
2344   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2345 }
2346 }

```

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.

```

2347 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2348 {
2349   \str_if_eq:nnTF { #3 } { s }
2350   { \@@_patch_preamble_vi_i:nnnn { #1 } { #4 } }
2351   { \@@_patch_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2352 }

```

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.

```

2353 \cs_new_protected:Npn \@@_patch_preamble_vi_i:nnnn #1 #2
2354 {
2355   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2356   \tl_gclear:N \g_@@_pre_cell_tl
2357   \tl_gput_right:Nn \g_@@_preamble_tl
2358   {
2359     > {
2360       \dim_set:Nn \l_@@_col_width_dim { #2 }
2361       \@@_cell_begin:w
2362       \str_set:Nn \l_@@_hpos_cell_str { c }
2363     }
2364     c
2365     < {
2366       \@@_cell_end_for_w_s:
2367       #1
2368       \@@_adjust_size_box:
2369       \box_use_drop:N \l_@@_cell_box
2370     }
2371   }
2372   \int_gincr:N \c@jCol
2373   \@@_patch_preamble_xi:n
2374 }

```

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

```

2375 \cs_new_protected:Npn \@@_patch_preamble_vi_ii:nnnn #1 #2 #3 #4
2376 {
2377   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2378   \tl_gclear:N \g_@@_pre_cell_tl
2379   \tl_gput_right:Nn \g_@@_preamble_tl
2380   {
2381     > {

```

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.

```

2382         \dim_set:Nn \l_@@_col_width_dim { #4 }
2383         \hbox_set:Nw \l_@@_cell_box
2384         \@@_cell_begin:w
2385         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2386     }
2387     c
2388     < {
2389         \@@_cell_end:
2390         \hbox_set_end:
2391         % The following line is probably pointless
2392         % \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2393         #1
2394         \@@_adjust_size_box:
2395         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2396     }
2397 }

```

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

```

2398     \int_gincr:N \c@jCol
2399     \@@_patch_preamble_xi:n
2400 }

```

```

2401 \cs_new_protected:Npn \@@_special_W:
2402 {
2403     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2404     { \@@_warning:n { W-warning } }
2405 }

```

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).

```

2406 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2407 {
2408     \str_if_eq:nnTF { #1 } { [ ] }
2409     { \@@_patch_preamble_vii_i:w [ ] }
2410     { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2411 }
2412 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2413 { \@@_patch_preamble_vii_ii:n { #1 } }
2414 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2415 {
2416     \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2417     {
2418         \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2419         \tl_gclear:N \g_@@_pre_cell_tl
2420         \tl_gput_right:Nn \g_@@_preamble_tl
2421         {
2422             > {
2423                 \@@_cell_begin:w
2424                 \keys_set:nn { siunitx } { #1 }
2425                 \siunitx_cell_begin:w
2426             }
2427             c
2428             < { \siunitx_cell_end: \@@_cell_end: }
2429         }

```

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

```

2430     \int_gincr:N \c@jCol
2431     \@@_patch_preamble_xi:n
2432 }
2433 { \@@_fatal:n { Version-of-siunitx-too-old } }
2434 }

```


For (, [and \{.

```

2435 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2436 {
2437   \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.

```

2438   \int_compare:nNnTF \c@jCol = \c_zero_int
2439   {
2440     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2441     {

```

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

```

2442       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2443       \tl_gset:Nn \g_@@_right_delim_tl { . }
2444       \@@_patch_preamble:n #2
2445     }
2446   {
2447     \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2448     \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2449   }
2450 }
2451 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2452 }
2453 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2454 {
2455   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2456   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }
2457   \tl_if_in:nnTF { ( [ \{ ) ] \} } \left \right } { #2 }
2458   {
2459     \@@_error:nn { delimiter~after~opening } { #2 }
2460     \@@_patch_preamble:n
2461   }
2462   { \@@_patch_preamble:n #2 }
2463 }

```

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}).

```

2464 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2465 {
2466   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2467   \tl_if_in:nnTF { ) ] \} } { #2 }
2468   { \@@_patch_preamble_ix_i:nnn #1 #2 }
2469   {
2470     \tl_if_eq:nnTF { \q_stop } { #2 }
2471     {
2472       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2473       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2474       {
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     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2483     { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2484     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2485     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } } \c_false_bool }
2486     \@@_patch_preamble:n #2

```

```

2487     }
2488   }
2489 }
2490 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2491 {
2492   \tl_if_eq:nnTF { \q_stop } { #3 }
2493   {
2494     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2495     {
2496       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2497       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2498       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2499       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2500     }
2501     {
2502       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2503       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2504       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2505       \@@_error:nn { double~closing~delimiter } { #2 }
2506     }
2507   }
2508   {
2509     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2510     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2511     \@@_error:nn { double~closing~delimiter } { #2 }
2512     \@@_patch_preamble:n #3
2513   }
2514 }

```

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.

```

2515 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2516 {
2517   \str_if_eq:nnTF { #1 } { [ ]
2518     { \@@_patch_preamble_x_i:w [ ]
2519       { \@@_patch_preamble_x_i:w [ ] #1 }
2520     }
2521   \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2522   { \@@_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).

```

2523 \keys_define:nn { WithArrows / X-column }
2524 { 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.

```

2525 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2526 {

```

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).

```

2527   \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).

```

2528   \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`.

```

2529 \int_zero_new:N \l_@@_weight_int
2530 \int_set:Nn \l_@@_weight_int { 1 }
2531 \tl_set:Nn \l_tmpa_tl { #1 }
2532 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2533 \@@_keys_p_column:V \l_tmpa_tl
2534 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2535 \int_compare:nNnT \l_@@_weight_int < 0
2536 {
2537   \@@_error_or_warning:n { negative~weight }
2538   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2539 }
2540 \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).

```

2541 \bool_if:NTF \l_@@_X_columns_aux_bool
2542 {
2543   \exp_args:Nnx
2544   \@@_patch_preamble_iv_iv:nn
2545   { \l_@@_weight_int \l_@@_X_columns_dim }
2546   { minipage }
2547 }
2548 {
2549   \tl_gput_right:Nn \g_@@_preamble_tl
2550   {
2551     > {
2552       \@@_cell_begin:w
2553       \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`.

```

2554 \NotEmpty

```

The following code will nullify the box of the cell.

```

2555 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2556 { \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`.

```

2557 \begin { minipage } { 5 cm } \arraybackslash
2558 }
2559 c
2560 < {
2561   \end { minipage }
2562   \@@_cell_end:
2563 }
2564 }
2565 \int_gincr:N \c@jCol
2566 \@@_patch_preamble_xi:n
2567 }
2568 }

```

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 `@{...}`.

```

2569 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2570 {

```

```

2571 \str_if_eq:nnTF { #1 } { < }
2572 \@@_patch_preamble_xiii:n
2573 {
2574   \str_if_eq:nnTF { #1 } { @ }
2575   \@@_patch_preamble_xv:n
2576   {
2577     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2578     {
2579       \tl_gput_right:Nn \g_@@_preamble_tl
2580       { ! { \skip_horizontal:N \arrayrulewidth } }
2581     }
2582     {
2583       \exp_args:NNx
2584       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2585       {
2586         \tl_gput_right:Nn \g_@@_preamble_tl
2587         { ! { \skip_horizontal:N \arrayrulewidth } }
2588       }
2589     }
2590     \@@_patch_preamble:n { #1 }
2591   }
2592 }
2593 }
2594 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2595 {
2596   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2597   \@@_patch_preamble_xi:n
2598 }

```

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.

```

2599 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2600 {
2601   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2602   {
2603     \tl_gput_right:Nn \g_@@_preamble_tl
2604     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2605   }
2606   {
2607     \exp_args:NNx
2608     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2609     {
2610       \tl_gput_right:Nn \g_@@_preamble_tl
2611       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2612     }
2613     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2614   }
2615   \@@_patch_preamble:n
2616 }
2617 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2618 {
2619   \group_begin:
2620   \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2621   \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2622   \@temptokena { #2 }
2623   \@tempswatrue
2624   \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2625   \tl_gclear:N \g_@@_preamble_tl
2626   \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
2627   \group_end:
2628   \tl_set_eq:NN #1 \g_@@_preamble_tl
2629 }

```

12 The redefinition of `\multicolumn`

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

```
2630 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2631 {
```

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`.

```
2632 \multispan { #1 }
2633 \begingroup
2634 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
2635 \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2636 \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
```

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

```
2637 \@temptokena = { #2 }
2638 \@tempswatrue
2639 \@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`.

```
2640 \tl_gclear:N \g_@@_preamble_tl
2641 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
```

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

```
2642 \exp_args:NV \mkpream \g_@@_preamble_tl
2643 \addtopreamble \empty
2644 \endgroup
```

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

```
2645 \int_compare:nNnT { #1 } > 1
2646 {
2647   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2648   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2649   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2650   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2651   {
2652     {
2653       \int_compare:nNnTF \c@jCol = 0
2654       { \int_eval:n { \c@iRow + 1 } }
2655       { \int_use:N \c@iRow }
2656     }
2657     { \int_eval:n { \c@jCol + 1 } }
2658     {
2659       \int_compare:nNnTF \c@jCol = 0
2660       { \int_eval:n { \c@iRow + 1 } }
2661       { \int_use:N \c@iRow }
2662     }
2663     { \int_eval:n { \c@jCol + #1 } }
2664     { } % for the name of the block
2665   }
2666 }
```

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

```
2667 \cs_set:Npn \@sharp { #3 }
2668 \@arstrut
2669 \@preamble
2670 \null
```

We add some lines.

```

2671 \int_gadd:Nn \c@jCol { #1 - 1 }
2672 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2673   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2674 \ignorespaces
2675 }

```

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`.

```

2676 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2677 {
2678   \str_case:nnF { #1 }
2679   {
2680     c { \@@_patch_m_preamble_i:n #1 }
2681     l { \@@_patch_m_preamble_i:n #1 }
2682     r { \@@_patch_m_preamble_i:n #1 }
2683     > { \@@_patch_m_preamble_ii:nn #1 }
2684     ! { \@@_patch_m_preamble_ii:nn #1 }
2685     @ { \@@_patch_m_preamble_ii:nn #1 }
2686     | { \@@_patch_m_preamble_iii:n #1 }
2687     p { \@@_patch_m_preamble_iv:nnn t #1 }
2688     m { \@@_patch_m_preamble_iv:nnn c #1 }
2689     b { \@@_patch_m_preamble_iv:nnn b #1 }
2690     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2691     \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2692     \q_stop { }
2693   }
2694   {
2695     \tl_if_eq:nnT { #1 } { S }
2696     { \@@_fatal:n { unknown~column~type~S } }
2697     { \@@_fatal:nn { unknown~column~type } { #1 } }
2698   }
2699 }

```

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

```

2700 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2701 {
2702   \tl_gput_right:Nn \g_@@_preamble_tl
2703   {
2704     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2705     #1
2706     < \@@_cell_end:
2707   }

```

We test for the presence of a `<`.

```

2708   \@@_patch_m_preamble_x:n
2709 }

```

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

```

2710 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2711 {
2712   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2713   \@@_patch_m_preamble:n
2714 }

```

For `|`

```

2715 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2716 {
2717   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2718   \@@_patch_m_preamble:n
2719 }

```

For p, m and b

```

2720 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2721 {
2722   \tl_gput_right:Nn \g_@@_preamble_tl
2723   {
2724     > {
2725       \@@_cell_begin:w
2726       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2727       \mode_leave_vertical:
2728       \arraybackslash
2729       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2730     }
2731     c
2732     < {
2733       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2734       \end { minipage }
2735       \@@_cell_end:
2736     }
2737   }

```

We test for the presence of a <.

```

2738   \@@_patch_m_preamble_x:n
2739 }

```

For w and W

```

2740 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2741 {
2742   \tl_gput_right:Nn \g_@@_preamble_tl
2743   {
2744     > {
2745       \dim_set:Nn \l_@@_col_width_dim { #4 }
2746       \hbox_set:Nw \l_@@_cell_box
2747       \@@_cell_begin:w
2748       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2749     }
2750     c
2751     < {
2752       \@@_cell_end:
2753       \hbox_set_end:
2754       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2755       #1
2756       \@@_adjust_size_box:
2757       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2758     }
2759   }

```

We test for the presence of a <.

```

2760   \@@_patch_m_preamble_x:n
2761 }

```

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

```

2762 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2763 {
2764   \str_if_eq:nnTF { #1 } { < }
2765   \@@_patch_m_preamble_ix:n
2766   { \@@_patch_m_preamble:n { #1 } }
2767 }
2768 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2769 {
2770   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2771   \@@_patch_m_preamble_x:n
2772 }

```

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`).

```

2773 \cs_new_protected:Npn \@@_put_box_in_flow:
2774 {
2775   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2776   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2777   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2778     { \box_use_drop:N \l_tmpa_box }
2779   \@@_put_box_in_flow_i:
2780 }

```

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).

```

2781 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2782 {
2783   \pgfpicture
2784     \@@_qpoint:n { row - 1 }
2785     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2786     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2787     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2788     \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).

```

2789   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2790     {
2791       \int_set:Nn \l_tmpa_int
2792       {
2793         \str_range:Nnn
2794           \l_@@_baseline_tl
2795           6
2796           { \tl_count:V \l_@@_baseline_tl }
2797       }
2798       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2799     }
2800     {
2801       \str_case:VnF \l_@@_baseline_tl
2802       {
2803         { t } { \int_set:Nn \l_tmpa_int 1 }
2804         { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2805       }
2806       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2807       \bool_lazy_or:nnT
2808         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2809         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2810       {
2811         \@@_error:n { bad~value~for~baseline }
2812         \int_set:Nn \l_tmpa_int 1
2813       }
2814       \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2815     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2816   }
2817   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2818   \endpgfpicture
2819   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2820   \box_use_drop:N \l_tmpa_box
2821 }

```


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).

```
2822 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2823 {
```

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.

```
2824 \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2825 {
2826   \box_set_wd:Nn \l_@@_the_array_box
2827   { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2828 }
```

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

```
2829 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2830 \bool_if:NT \l_@@_caption_above_bool
2831 {
2832   \tl_if_empty:NF \l_@@_caption_tl
2833   {
2834     \bool_set_false:N \g_@@_caption_finished_bool
2835     \int_gzero:N \c@tabularnote
2836     \@@_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).

```
2837     \int_compare:nNnT \g_@@_notes_caption_int > 0
2838     {
2839       \tl_gput_right:Nx \g_@@_aux_tl
2840       {
2841         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2842         { \int_use:N \g_@@_notes_caption_int }
2843       }
2844       \int_gzero:N \g_@@_notes_caption_int
2845     }
2846   }
2847 }
```

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

```
2848 \hbox
2849 {
2850   \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.

```
2851 \@@_create_extra_nodes:
2852 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2853 }
```

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

```
2854 \bool_lazy_any:nT
2855 {
2856   { ! \seq_if_empty_p:N \g_@@_notes_seq }
2857   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2858   { ! \tl_if_empty_p:N \g_@@_tabularnote_tl }
2859 }
```

```

2860 \@@_insert_tabularnotes:
2861 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2862 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2863 \end { minipage }
2864 }

```

```

2865 \cs_new_protected:Npn \@@_insert_caption:
2866 {
2867 \tl_if_empty:NF \l_@@_caption_tl
2868 {
2869 \cs_if_exist:NTF \@capttype
2870 { \@@_insert_caption_i: }
2871 { \@@_error:n { caption-outside~float } }
2872 }
2873 }

```

```

2874 \cs_new_protected:Npn \@@_insert_caption_i:
2875 {
2876 \group_begin:

```

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

```

2877 \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.

```

2878 \IfPackageLoadedTF { floatrow }
2879 { \cs_set_eq:NN \@makecaption \FR@makecaption }
2880 { }
2881 \tl_if_empty:NTF \l_@@_short_caption_tl
2882 { \caption }
2883 { \caption [ \l_@@_short_caption_tl ] }
2884 { \l_@@_caption_tl }
2885 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2886 \group_end:
2887 }

```

```

2888 \cs_new_protected:Npn \@@_tabularnote_error:n #1
2889 {
2890 \@@_error_or_warning:n { tabularnote~below~the~tabular }
2891 \@@_gredirect_none:n { tabularnote~below~the~tabular }
2892 }

```

```

2893 \cs_new_protected:Npn \@@_insert_tabularnotes:
2894 {
2895 \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2896 \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2897 \skip_vertical:N 0.65ex

```

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

```

2898 \group_begin:
2899 \l_@@_notes_code_before_tl
2900 \tl_if_empty:NF \g_@@_tabularnote_tl
2901 {
2902 \g_@@_tabularnote_tl \par
2903 \tl_gclear:N \g_@@_tabularnote_tl
2904 }

```

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.

```

2905 \int_compare:nNnT \c@tabularnote > 0
2906 {
2907 \bool_if:NTF \l_@@_notes_para_bool

```

```

2908     {
2909         \begin { tabularnotes* }
2910         \seq_map_inline:Nn \g_@@_notes_seq
2911         { \@@_one_tabularnote:nn ##1 }
2912         \strut
2913     \end { tabularnotes* }

```

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

```

2914     \par
2915 }
2916 {
2917     \tabularnotes
2918     \seq_map_inline:Nn \g_@@_notes_seq
2919     { \@@_one_tabularnote:nn ##1 }
2920     \strut
2921     \endtabularnotes
2922 }
2923 }
2924 \unskip
2925 \group_end:
2926 \bool_if:NT \l_@@_notes_bottomrule_bool
2927 {
2928     \IfPackageLoadedTF { booktabs }
2929     {

```

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

```

2930         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2931         { \CT@arc@ \hrule height \heavyrulewidth }
2932     }
2933     { \@@_error_or_warning:n { bottomrule-without~booktabs } }
2934 }
2935 \l_@@_notes_code_after_tl
2936 \seq_gclear:N \g_@@_notes_seq
2937 \seq_gclear:N \g_@@_notes_in_caption_seq
2938 \int_gzero:N \c@tabularnote
2939 }

```

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 currfication.

```

2940 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
2941 {
2942     \tl_if_novalue:nTF { #1 }
2943     { \item }
2944     { \item [ \@@_notes_label_in_list:n { #1 } ] }
2945 }

```

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`.

```

2946 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2947 {
2948     \pgfpicture
2949     \@@_qpoint:n { row - 1 }
2950     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2951     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2952     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2953     \endpgfpicture
2954     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2955     \int_compare:nNnT \l_@@_first_row_int = 0

```

```

2956 {
2957   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2958   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2959 }
2960 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2961 }

```

Now, the general case.

```

2962 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2963 {

```

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

```

2964   \tl_if_eq:NnT \l_@@_baseline_tl { t }
2965   { \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`.

```

2966   \pgfpicture
2967   \@@_qpoint:n { row - 1 }
2968   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2969   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2970   {
2971     \int_set:Nn \l_tmpa_int
2972     {
2973       \str_range:Nnn
2974       \l_@@_baseline_tl
2975       6
2976       { \tl_count:V \l_@@_baseline_tl }
2977     }
2978     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2979   }
2980   {
2981     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2982     \bool_lazy_or:nnT
2983     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2984     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2985     {
2986       \@@_error:n { bad~value~for~baseline }
2987       \int_set:Nn \l_tmpa_int 1
2988     }
2989     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2990   }
2991   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2992   \endpgfpicture
2993   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2994   \int_compare:nNnT \l_@@_first_row_int = 0
2995   {
2996     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2997     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2998   }
2999   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3000 }

```

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.

```

3001 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3002 {

```

We will compute the real width of both delimiters used.

```

3003   \dim_zero_new:N \l_@@_real_left_delim_dim
3004   \dim_zero_new:N \l_@@_real_right_delim_dim
3005   \hbox_set:Nn \l_tmpb_box
3006   {

```

```

3007     \c_math_toggle_token
3008     \left #1
3009     \vcenter
3010     {
3011         \vbox_to_ht:nn
3012         { \box_ht_plus_dp:N \l_tmpa_box }
3013         { }
3014     }
3015     \right .
3016     \c_math_toggle_token
3017 }
3018 \dim_set:Nn \l_@@_real_left_delim_dim
3019 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
3020 \hbox_set:Nn \l_tmpb_box
3021 {
3022     \c_math_toggle_token
3023     \left .
3024     \vbox_to_ht:nn
3025     { \box_ht_plus_dp:N \l_tmpa_box }
3026     { }
3027     \right #2
3028     \c_math_toggle_token
3029 }
3030 \dim_set:Nn \l_@@_real_right_delim_dim
3031 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }

```

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

```

3032     \skip_horizontal:N \l_@@_left_delim_dim
3033     \skip_horizontal:N -\l_@@_real_left_delim_dim
3034     \@@_put_box_in_flow:
3035     \skip_horizontal:N \l_@@_right_delim_dim
3036     \skip_horizontal:N -\l_@@_real_right_delim_dim
3037 }

```

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).

```

3038 \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).

```

3039 {
3040     \peek_remove_spaces:n
3041     {
3042         \peek_meaning:NTF \end
3043         \@@_analyze_end:Nn
3044         {
3045             \@@_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`).

```

3046         \@@_array:V \g_@@_preamble_tl
3047     }
3048 }
3049 }
3050 {
3051     \@@_create_col_nodes:
3052     \endarray
3053 }

```

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

```
3054 \NewDocumentEnvironment { @@-light-syntax } { b }
3055 {
```

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`.

```
3056 \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3057 \tl_map_inline:nn { #1 }
3058 {
3059   \str_if_eq:nnT { ##1 } { & }
3060   { \@@_fatal:n { ampersand-in~light-syntax } }
3061   \str_if_eq:nnT { ##1 } { \ }
3062   { \@@_fatal:n { double-backslash-in~light-syntax } }
3063 }
```

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`.

```
3064 \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3065 }
```

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.

```
3066 {
3067   \@@_create_col_nodes:
3068   \endarray
3069 }

3070 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3071 {
3072   \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).

```
3073 \seq_clear_new:N \l_@@_rows_seq
```

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

```
3074 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3075 \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3076 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3077 \tl_if_empty:NF \l_tmpa_tl
3078 { \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.

```
3079 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3080 { \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`).

```
3081 \tl_clear_new:N \l_@@_new_body_tl
3082 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3083 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3084 \@@_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).

```
3085 \seq_map_inline:Nn \l_@@_rows_seq
3086 {
3087   \tl_put_right:Nn \l_@@_new_body_tl { \ }
3088   \@@_line_with_light_syntax:n { ##1 }
3089 }
3090 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3091 {
3092   \int_set:Nn \l_@@_last_col_int
3093   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3094 }
```

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.

```
3095 \@@_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`).

```
3096 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3097 }
3098 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3099 {
3100   \seq_clear_new:N \l_@@_cells_seq
3101   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3102   \int_set:Nn \l_@@_nb_cols_int
3103   {
3104     \int_max:nn
3105     \l_@@_nb_cols_int
3106     { \seq_count:N \l_@@_cells_seq }
3107   }
3108   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3109   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3110   \seq_map_inline:Nn \l_@@_cells_seq
3111   { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3112 }
3113 \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`.

```
3114 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3115 {
3116   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3117   { \@@_fatal:n { empty-environment } }
```

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

```
3118 \end { #2 }
3119 }
```

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).

```
3120 \cs_new:Npn \@@_create_col_nodes:
3121 {
3122   \crrc
3123   \int_compare:nNnT \l_@@_first_col_int = 0
3124   {
```

```

3125 \omit
3126 \hbox_overlap_left:n
3127 {
3128     \bool_if:NT \l_@@_code_before_bool
3129     { \pgfsys@markposition { \@@_env: - col - 0 } }
3130     \pgfpicture
3131     \pgfrememberpicturepositiononpagetrue
3132     \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3133     \str_if_empty:NF \l_@@_name_str
3134     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3135     \endpgfpicture
3136     \skip_horizontal:N 2\col@sep
3137     \skip_horizontal:N \g_@@_width_first_col_dim
3138 }
3139 &
3140 }
3141 \omit

```

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

```

3142 \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`).

```

3143 \int_compare:nNnTF \l_@@_first_col_int = 0
3144 {
3145     \bool_if:NT \l_@@_code_before_bool
3146     {
3147         \hbox
3148         {
3149             \skip_horizontal:N -0.5\arrayrulewidth
3150             \pgfsys@markposition { \@@_env: - col - 1 }
3151             \skip_horizontal:N 0.5\arrayrulewidth
3152         }
3153     }
3154     \pgfpicture
3155     \pgfrememberpicturepositiononpagetrue
3156     \pgfcoordinate { \@@_env: - col - 1 }
3157     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3158     \str_if_empty:NF \l_@@_name_str
3159     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3160     \endpgfpicture
3161 }
3162 {
3163     \bool_if:NT \l_@@_code_before_bool
3164     {
3165         \hbox
3166         {
3167             \skip_horizontal:N 0.5\arrayrulewidth
3168             \pgfsys@markposition { \@@_env: - col - 1 }
3169             \skip_horizontal:N -0.5\arrayrulewidth
3170         }
3171     }
3172     \pgfpicture
3173     \pgfrememberpicturepositiononpagetrue
3174     \pgfcoordinate { \@@_env: - col - 1 }
3175     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3176     \str_if_empty:NF \l_@@_name_str
3177     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3178     \endpgfpicture
3179 }

```

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.

```

3180 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3181 \bool_if:NF \l_@@_auto_columns_width_bool
3182 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3183 {
3184   \bool_lazy_and:nnTF
3185     \l_@@_auto_columns_width_bool
3186     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3187     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3188     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3189     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3190 }
3191 \skip_horizontal:N \g_tmpa_skip
3192 \hbox
3193 {
3194   \bool_if:NT \l_@@_code_before_bool
3195   {
3196     \hbox
3197     {
3198       \skip_horizontal:N -0.5\arrayrulewidth
3199       \pgfsys@markposition { \@@_env: - col - 2 }
3200       \skip_horizontal:N 0.5\arrayrulewidth
3201     }
3202   }
3203   \pgfpicture
3204   \pgfrememberpicturerepositiononpagetrue
3205   \pgfcoordinate { \@@_env: - col - 2 }
3206   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3207   \str_if_empty:NF \l_@@_name_str
3208   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3209   \endpgfpicture
3210 }

```

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.

```

3211 \int_gset:Nn \g_tmpa_int 1
3212 \bool_if:NTF \g_@@_last_col_found_bool
3213 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3214 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3215 {
3216   &
3217   \omit
3218   \int_gincr:N \g_tmpa_int

```

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

```

3219   \skip_horizontal:N \g_tmpa_skip
3220   \bool_if:NT \l_@@_code_before_bool
3221   {
3222     \hbox
3223     {
3224       \skip_horizontal:N -0.5\arrayrulewidth
3225       \pgfsys@markposition
3226       { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3227       \skip_horizontal:N 0.5\arrayrulewidth
3228     }
3229   }

```

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

```

3230   \pgfpicture
3231   \pgfrememberpicturerepositiononpagetrue
3232   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3233   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3234   \str_if_empty:NF \l_@@_name_str

```

```

3235         {
3236             \pgfnodealias
3237             { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3238             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3239         }
3240     \endpgfpicture
3241 }

```

```

3242 &
3243 \omit

```

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

```

3244 \int_compare:nNnT \g_@@_col_total_int = 1
3245 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3246 \skip_horizontal:N \g_tmpa_skip
3247 \int_gincr:N \g_tmpa_int
3248 \bool_lazy_all:nT
3249 {
3250     \g_@@_NiceArray_bool
3251     { \bool_not_p:n \l_@@_NiceTabular_bool }
3252     { \clist_if_empty_p:N \l_@@_vlines_clist }
3253     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3254     { ! \l_@@_bar_at_end_of_pream_bool }
3255 }
3256 { \skip_horizontal:N -\col@sep }
3257 \bool_if:NT \l_@@_code_before_bool
3258 {
3259     \hbox
3260     {
3261         \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.

```

3262         \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3263         { \skip_horizontal:N -\arraycolsep }
3264     \pgfsys@markposition
3265     { \@@_env: - col - \int_eval:n {
3266         \g_tmpa_int + 1 } }
3267     \skip_horizontal:N 0.5\arrayrulewidth
3268     \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3269     { \skip_horizontal:N \arraycolsep }
3270 }
3271 }
3272 \pgfpicture
3273 \pgfrememberpicturepositiononpagetrue
3274 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3275 {
3276     \bool_lazy_and:nnTF \l_@@_Matrix_bool \g_@@_NiceArray_bool
3277     {
3278         \pgfpoint
3279         { - 0.5 \arrayrulewidth - \arraycolsep }
3280         \c_zero_dim
3281     }
3282     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3283 }
3284 \str_if_empty:NF \l_@@_name_str
3285 {
3286     \pgfnodealias
3287     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3288     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3289 }
3290 \endpgfpicture

```

```

3291 \bool_if:NT \g_@@_last_col_found_bool
3292 {
3293   \hbox_overlap_right:n
3294   {
3295     \skip_horizontal:N \g_@@_width_last_col_dim
3296     \bool_if:NT \l_@@_code_before_bool
3297     {
3298       \pgfsys@markposition
3299       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3300     }
3301     \pgfpicture
3302     \pgfrememberpicturepositiononpagetrue
3303     \pgfcoordinate
3304     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3305     \pgfpointorigin
3306     \str_if_empty:NF \l_@@_name_str
3307     {
3308       \pgfnodealias
3309       {
3310         \l_@@_name_str - col
3311         - \int_eval:n { \g_@@_col_total_int + 1 }
3312       }
3313       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3314     }
3315     \endpgfpicture
3316   }
3317 }
3318 \cr
3319 }

```

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

```

3320 \tl_const:Nn \c_@@_preamble_first_col_tl
3321 {
3322   >
3323   {

```

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).

```

3324 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3325 \bool_gset_true:N \g_@@_after_col_zero_bool
3326 \@@_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.

```

3327 \hbox_set:Nw \l_@@_cell_box
3328 \@@_math_toggle_token:
3329 \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”.

```

3330 \bool_lazy_and:nnT
3331 { \int_compare_p:nNn \c@iRow > 0 }
3332 {
3333   \bool_lazy_or_p:nn
3334   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3335   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3336 }
3337 {
3338   \l_@@_code_for_first_col_tl
3339   \xglobal \colorlet { nicematrix-first-col } { . }
3340 }
3341 }

```

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`.

```

3342   l
3343   <
3344   {
3345     \@@_math_toggle_token:
3346     \hbox_set_end:
3347     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3348     \@@_adjust_size_box:
3349     \@@_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.

```

3350     \dim_gset:Nn \g_@@_width_first_col_dim
3351     { \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.

```

3352     \hbox_overlap_left:n
3353     {
3354       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3355       \@@_node_for_cell:
3356       { \box_use_drop:N \l_@@_cell_box }
3357       \skip_horizontal:N \l_@@_left_delim_dim
3358       \skip_horizontal:N \l_@@_left_margin_dim
3359       \skip_horizontal:N \l_@@_extra_left_margin_dim
3360     }
3361     \bool_gset_false:N \g_@@_empty_cell_bool
3362     \skip_horizontal:N -2\col@sep
3363   }
3364 }

```

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

```

3365 \tl_const:Nn \c_@@_preamble_last_col_tl
3366 {
3367   >
3368   {

```

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).

```

3369     \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.

```

3370     \bool_gset_true:N \g_@@_last_col_found_bool
3371     \int_gincr:N \c@jCol
3372     \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.

```

3373     \hbox_set:Nw \l_@@_cell_box
3374     \@@_math_toggle_token:
3375     \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”.

```

3376     \int_compare:nNnT \c@iRow > 0
3377     {
3378       \bool_lazy_or:nnT
3379       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3380       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3381       {
3382         \l_@@_code_for_last_col_tl
3383         \xglobal \colorlet { nicematrix-last-col } { . }
3384       }
3385     }
3386   }
3387   l

```

```

3388 <
3389 {
3390   \@@_math_toggle_token:
3391   \hbox_set_end:
3392   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3393   \@@_adjust_size_box:
3394   \@@_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.

```

3395   \dim_gset:Nn \g_@@_width_last_col_dim
3396   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3397   \skip_horizontal:N -2\col@sep

```

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

```

3398   \hbox_overlap_right:n
3399   {
3400     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3401     {
3402       \skip_horizontal:N \l_@@_right_delim_dim
3403       \skip_horizontal:N \l_@@_right_margin_dim
3404       \skip_horizontal:N \l_@@_extra_right_margin_dim
3405       \@@_node_for_cell:
3406     }
3407   }
3408   \bool_gset_false:N \g_@@_empty_cell_bool
3409 }
3410 }

```

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.

```

3411 \NewDocumentEnvironment { NiceArray } { }
3412 {
3413   \bool_gset_true:N \g_@@_NiceArray_bool
3414   \str_if_empty:NT \g_@@_name_env_str
3415   { \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).

```

3416   \NiceArrayWithDelims . .
3417 }
3418 { \endNiceArrayWithDelims }

```

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

```

3419 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3420 {
3421   \NewDocumentEnvironment { #1 NiceArray } { }
3422   {
3423     \bool_gset_false:N \g_@@_NiceArray_bool
3424     \str_if_empty:NT \g_@@_name_env_str
3425     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3426     \@@_test_if_math_mode:
3427     \NiceArrayWithDelims #2 #3
3428   }
3429   { \endNiceArrayWithDelims }
3430 }
3431 \@@_def_env:nnn p ( )
3432 \@@_def_env:nnn b [ ]
3433 \@@_def_env:nnn B \{ \}
3434 \@@_def_env:nnn v | |
3435 \@@_def_env:nnn V \| \|

```

13 The environment {NiceMatrix} and its variants

```

3436 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3437 {
3438   \bool_set_true:N \l_@@_Matrix_bool
3439   \use:c { #1 NiceArray }
3440   {
3441     *
3442     {
3443       \int_case:nnF \l_@@_last_col_int
3444       {
3445         { -2 } { \c@MaxMatrixCols }
3446         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3447       }
3448       { \int_eval:n { \l_@@_last_col_int - 1 } }
3449     }
3450     { #2 }
3451   }
3452 }
3453 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3454 \clist_map_inline:nn { p , b , B , v , V }
3455 {
3456   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3457   {
3458     \bool_gset_false:N \g_@@_NiceArray_bool
3459     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3460     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3461     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3462   }
3463   { \use:c { end #1 NiceArray } }
3464 }

```

We define also an environment {NiceMatrix}

```

3465 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3466 {
3467   \bool_gset_false:N \g_@@_NiceArray_bool
3468   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3469   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3470   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3471 }
3472 { \endNiceArray }

```

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

```

3473 \cs_new_protected:Npn \@@_NotEmpty:
3474 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3475 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3476 {

```

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.

```

3477   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3478   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3479   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3480   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3481   \int_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim

```

```

3482 {
3483   \bool_if:NT \l_@@_hvlines_bool
3484   {
3485     \bool_set_true:N \l_@@_except_borders_bool
3486     % we should try to be more efficient in the number of lines of code here
3487     \tl_if_empty:NTF \l_@@_rules_color_tl
3488     {
3489       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3490       {
3491         \@@_stroke_block:nnn
3492         { rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim }
3493         { 1-1 }
3494         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3495       }
3496     }
3497     {
3498       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3499       {
3500         \@@_stroke_block:nnn
3501         {
3502           rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3503           draw = \l_@@_rules_color_tl
3504         }
3505         { 1-1 }
3506         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3507       }
3508     }
3509   }
3510 }
3511 \tl_if_empty:NF \l_@@_short_caption_tl
3512 {
3513   \tl_if_empty:NT \l_@@_caption_tl
3514   {
3515     \@@_error_or_warning:n { short-caption~without~caption }
3516     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3517   }
3518 }
3519 \tl_if_empty:NF \l_@@_label_tl
3520 {
3521   \tl_if_empty:NT \l_@@_caption_tl
3522   { \@@_error_or_warning:n { label~without~caption } }
3523 }
3524 \NewDocumentEnvironment { TabularNote } { b }
3525 {
3526   \bool_if:NTF \l_@@_in_code_after_bool
3527   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3528   {
3529     \tl_if_empty:NF \g_@@_tabularnote_tl
3530     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3531     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3532   }
3533 }
3534 { }
3535 \bool_set_true:N \l_@@_NiceTabular_bool
3536 \NiceArray { #2 }
3537 }
3538 { \endNiceArray }
3539 \cs_set_protected:Npn \@@_newcolumnntype #1
3540 {
3541   \cs_if_free:cT { NC @ find @ #1 }
3542   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3543   \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }

```

```

3544 \peek_meaning:NTF [
3545   { \newcol@ #1 }
3546   { \newcol@ #1 [ 0 ] }
3547 ]

```

```

3548 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3549 {

```

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}`).

```

3550 \IfPackageLoadedTF { tabularx }
3551   { \newcolumnntype { X } { @@_X } }
3552   { }
3553 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3554 \dim_zero_new:N \l_@@_width_dim
3555 \dim_set:Nn \l_@@_width_dim { #1 }
3556 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3557 \bool_set_true:N \l_@@_NiceTabular_bool
3558 \NiceArray { #3 }
3559 }
3560 { \endNiceArray }

```

```

3561 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3562 {
3563   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3564   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3565   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3566   \bool_set_true:N \l_@@_NiceTabular_bool
3567   \NiceArray { #3 }
3568 }
3569 { \endNiceArray }

```

15 After the construction of the array

```

3570 \cs_new_protected:Npn \@@_after_array:
3571 {
3572   \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.

```

3573 \bool_if:NT \g_@@_last_col_found_bool
3574   { \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`.

```

3575 \bool_if:NT \l_@@_last_col_without_value_bool
3576   { \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.

```

3577 \bool_if:NT \l_@@_last_row_without_value_bool
3578   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3579 \tl_gput_right:Nx \g_@@_aux_tl
3580 {
3581   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3582   {
3583     \int_use:N \l_@@_first_row_int ,
3584     \int_use:N \c@iRow ,

```



```

3585         \int_use:N \g_@@_row_total_int ,
3586         \int_use:N \l_@@_first_col_int ,
3587         \int_use:N \c@jCol ,
3588         \int_use:N \g_@@_col_total_int
3589     }
3590 }

```

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`).

```

3591 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3592 {
3593     \tl_gput_right:Nx \g_@@_aux_tl
3594     {
3595         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3596         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3597     }
3598 }
3599 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3600 {
3601     \tl_gput_right:Nx \g_@@_aux_tl
3602     {
3603         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3604         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3605         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3606         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3607     }
3608 }

```

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

```

3609 \@@_create_diag_nodes:

```

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

```

3610 \pgfpicture
3611 \int_step_inline:nn \c@iRow
3612 {
3613     \pgfnodealias
3614     { \@@_env: - ##1 - last }
3615     { \@@_env: - ##1 - \int_use:N \c@jCol }
3616 }
3617 \int_step_inline:nn \c@jCol
3618 {
3619     \pgfnodealias
3620     { \@@_env: - last - ##1 }
3621     { \@@_env: - \int_use:N \c@iRow - ##1 }
3622 }
3623 \str_if_empty:NF \l_@@_name_str
3624 {
3625     \int_step_inline:nn \c@iRow
3626     {
3627         \pgfnodealias
3628         { \l_@@_name_str - ##1 - last }
3629         { \@@_env: - ##1 - \int_use:N \c@jCol }
3630     }
3631     \int_step_inline:nn \c@jCol
3632     {
3633         \pgfnodealias
3634         { \l_@@_name_str - last - ##1 }
3635         { \@@_env: - \int_use:N \c@iRow - ##1 }
3636     }
3637 }
3638 \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.

```

3639   \bool_if:NT \l_@@_parallelize_diags_bool
3640   {
3641     \int_gzero_new:N \g_@@_ddots_int
3642     \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.

```

3643     \dim_gzero_new:N \g_@@_delta_x_one_dim
3644     \dim_gzero_new:N \g_@@_delta_y_one_dim
3645     \dim_gzero_new:N \g_@@_delta_x_two_dim
3646     \dim_gzero_new:N \g_@@_delta_y_two_dim
3647   }
3648   \int_zero_new:N \l_@@_initial_i_int
3649   \int_zero_new:N \l_@@_initial_j_int
3650   \int_zero_new:N \l_@@_final_i_int
3651   \int_zero_new:N \l_@@_final_j_int
3652   \bool_set_false:N \l_@@_initial_open_bool
3653   \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.

```

3654   \bool_if:NT \l_@@_small_bool
3655   {
3656     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3657     \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.

```

3658     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3659     { 0.6 \l_@@_xdots_shorten_start_dim }
3660     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3661     { 0.6 \l_@@_xdots_shorten_end_dim }
3662   }

```

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

```

3663   \@@_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.

```

3664   \@@_compute_corners:

```

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

```

3665   \@@_adjust_pos_of_blocks_seq:
3666   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3667   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3668   \IfPackageLoadedTF { tikz }
3669   {
3670     \tikzset
3671     {

```

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

```

3672         every-picture / .style =
3673         {
3674             overlay ,
3675             remember-picture ,
3676             name-prefix = \@@_env: -
3677         }
3678     }
3679 }
3680 { }
3681 \cs_set_eq:NN \ialign \@@_old_ialign:
3682 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3683 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3684 \cs_set_eq:NN \OverBrace \@@_OverBrace
3685 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3686 \cs_set_eq:NN \line \@@_line
3687 \g_@@_pre_code_after_tl
3688 \tl_gclear:N \g_@@_pre_code_after_tl

```

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

```

3689 \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).

```

3690 \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`).

```

3691 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3692 { \@@_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:`.

```

3693 \bool_set_true:N \l_@@_in_code_after_bool
3694 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3695 \scan_stop:
3696 \tl_gclear:N \g_nicematrix_code_after_tl
3697 \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.

```

3698 \tl_if_empty:NF \g_@@_pre_code_before_tl
3699 {
3700     \tl_gput_right:Nx \g_@@_aux_tl
3701     {
3702         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3703         { \exp_not:V \g_@@_pre_code_before_tl }
3704     }
3705     \tl_gclear:N \g_@@_pre_code_before_tl
3706 }
3707 \tl_if_empty:NF \g_nicematrix_code_before_tl
3708 {
3709     \tl_gput_right:Nx \g_@@_aux_tl
3710     {
3711         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3712         { \exp_not:V \g_nicematrix_code_before_tl }
3713     }
3714     \tl_gclear:N \g_nicematrix_code_before_tl
3715 }
3716 \str_gclear:N \g_@@_name_env_str

```

```
3717 \@@_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`.

```
3718 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3719 }
```

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`.

```
3720 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3721 { \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).

```
3722 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3723 {
3724   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3725   { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3726 }
```

The following command must *not* be protected.

```
3727 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3728 {
3729   { #1 }
3730   { #2 }
3731   {
3732     \int_compare:nNnTF { #3 } > { 99 }
3733     { \int_use:N \c@iRow }
3734     { #3 }
3735   }
3736   {
3737     \int_compare:nNnTF { #4 } > { 99 }
3738     { \int_use:N \c@jCol }
3739     { #4 }
3740   }
3741   { #5 }
3742 }
```

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).

```
3743 \hook_gput_code:nnn { begindocument } { . }
3744 {
3745   \cs_new_protected:Npx \@@_draw_dotted_lines:
3746   {
3747     \c_@@_pgfortikzpicture_tl
3748     \@@_draw_dotted_lines_i:
3749     \c_@@_endpgfortikzpicture_tl
3750   }
3751 }
```

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:`.

```

3752 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3753 {
3754   \pgfrememberpicturepositiononpagetrue
3755   \pgf@relevantforpicturesizefalse
3756   \g_@@_HVdotsfor_lines_tl
3757   \g_@@_Vdots_lines_tl
3758   \g_@@_Ddots_lines_tl
3759   \g_@@_Iddots_lines_tl
3760   \g_@@_Cdots_lines_tl
3761   \g_@@_Ldots_lines_tl
3762 }

3763 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3764 {
3765   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3766   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3767 }

```

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

```

3768 \pgfdeclareshape { @@_diag_node }
3769 {
3770   \savedanchor { \five }
3771   {
3772     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3773     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3774   }
3775   \anchor { 5 } { \five }
3776   \anchor { center } { \pgfpointorigin }
3777 }

```

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).

```

3778 \cs_new_protected:Npn \@@_create_diag_nodes:
3779 {
3780   \pgfpicture
3781   \pgfrememberpicturepositiononpagetrue
3782   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3783   {
3784     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3785     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3786     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3787     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3788     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3789     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3790     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3791     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3792     \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.

```

3793   \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3794   \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3795   \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3796   \str_if_empty:NF \l_@@_name_str
3797   { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3798 }

```

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

```

3799   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }

```

```

3800 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3801 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3802 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3803 \pgfcoordinate
3804 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3805 \pgfnodealias
3806 { \@@_env: - last }
3807 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3808 \str_if_empty:NF \l_@@_name_str
3809 {
3810   \pgfnodealias
3811   { \l_@@_name_str - \int_use:N \l_tmpa_int }
3812   { \@@_env: - \int_use:N \l_tmpa_int }
3813   \pgfnodealias
3814   { \l_@@_name_str - last }
3815   { \@@_env: - last }
3816 }
3817 \endpgfpicture
3818 }

```

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.

```

3819 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3820 {

```

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

```

3821 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

3822 \int_set:Nn \l_@@_initial_i_int { #1 }
3823 \int_set:Nn \l_@@_initial_j_int { #2 }
3824 \int_set:Nn \l_@@_final_i_int { #1 }
3825 \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.

```

3826     \bool_set_false:N \l_@@_stop_loop_bool
3827     \bool_do_until:Nn \l_@@_stop_loop_bool
3828     {
3829         \int_add:Nn \l_@@_final_i_int { #3 }
3830         \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3831     \bool_set_false:N \l_@@_final_open_bool
3832     \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3833     {
3834         \int_compare:nNnTF { #3 } = 1
3835         { \bool_set_true:N \l_@@_final_open_bool }
3836         {
3837             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3838             { \bool_set_true:N \l_@@_final_open_bool }
3839         }
3840     }
3841     {
3842         \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3843         {
3844             \int_compare:nNnT { #4 } = { -1 }
3845             { \bool_set_true:N \l_@@_final_open_bool }
3846         }
3847         {
3848             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3849             {
3850                 \int_compare:nNnT { #4 } = 1
3851                 { \bool_set_true:N \l_@@_final_open_bool }
3852             }
3853         }
3854     }
3855     \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.

```

3856     {

```

We do a step backwards.

```

3857         \int_sub:Nn \l_@@_final_i_int { #3 }
3858         \int_sub:Nn \l_@@_final_j_int { #4 }
3859         \bool_set_true:N \l_@@_stop_loop_bool
3860     }

```

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`.

```

3861     {
3862         \cs_if_exist:cTF
3863         {
3864             @@ _ dotted _
3865             \int_use:N \l_@@_final_i_int -
3866             \int_use:N \l_@@_final_j_int
3867         }
3868         {
3869             \int_sub:Nn \l_@@_final_i_int { #3 }
3870             \int_sub:Nn \l_@@_final_j_int { #4 }
3871             \bool_set_true:N \l_@@_final_open_bool
3872             \bool_set_true:N \l_@@_stop_loop_bool
3873         }
3874         {
3875             \cs_if_exist:cTF
3876             {
3877                 pgf @ sh @ ns @ \@@_env:
3878                 - \int_use:N \l_@@_final_i_int

```

```

3879         - \int_use:N \l_@@_final_j_int
3880     }
3881     { \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.

```

3882     {
3883         \cs_set:cpn
3884         {
3885             @@ _ dotted _
3886             \int_use:N \l_@@_final_i_int -
3887             \int_use:N \l_@@_final_j_int
3888         }
3889         { }
3890     }
3891 }
3892 }
3893 }

```

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

```

3894     \bool_set_false:N \l_@@_stop_loop_bool
3895     \bool_do_until:Nn \l_@@_stop_loop_bool
3896     {
3897         \int_sub:Nn \l_@@_initial_i_int { #3 }
3898         \int_sub:Nn \l_@@_initial_j_int { #4 }
3899         \bool_set_false:N \l_@@_initial_open_bool
3900         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3901         {
3902             \int_compare:nNnTF { #3 } = 1
3903             { \bool_set_true:N \l_@@_initial_open_bool }
3904             {
3905                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3906                 { \bool_set_true:N \l_@@_initial_open_bool }
3907             }
3908         }
3909         {
3910             \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3911             {
3912                 \int_compare:nNnT { #4 } = 1
3913                 { \bool_set_true:N \l_@@_initial_open_bool }
3914             }
3915             {
3916                 \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3917                 {
3918                     \int_compare:nNnT { #4 } = { -1 }
3919                     { \bool_set_true:N \l_@@_initial_open_bool }
3920                 }
3921             }
3922         }
3923         \bool_if:NnTF \l_@@_initial_open_bool
3924         {
3925             \int_add:Nn \l_@@_initial_i_int { #3 }
3926             \int_add:Nn \l_@@_initial_j_int { #4 }
3927             \bool_set_true:N \l_@@_stop_loop_bool
3928         }
3929         {
3930             \cs_if_exist:cTF
3931             {

```



```

3932         @@ _ dotted _
3933         \int_use:N \l_@@_initial_i_int -
3934         \int_use:N \l_@@_initial_j_int
3935     }
3936     {
3937         \int_add:Nn \l_@@_initial_i_int { #3 }
3938         \int_add:Nn \l_@@_initial_j_int { #4 }
3939         \bool_set_true:N \l_@@_initial_open_bool
3940         \bool_set_true:N \l_@@_stop_loop_bool
3941     }
3942     {
3943         \cs_if_exist:cTF
3944         {
3945             pgf @ sh @ ns @ \@@_env:
3946             - \int_use:N \l_@@_initial_i_int
3947             - \int_use:N \l_@@_initial_j_int
3948         }
3949         { \bool_set_true:N \l_@@_stop_loop_bool }
3950         {
3951             \cs_set:cpn
3952             {
3953                 @@ _ dotted _
3954                 \int_use:N \l_@@_initial_i_int -
3955                 \int_use:N \l_@@_initial_j_int
3956             }
3957             { }
3958         }
3959     }
3960 }
3961 }

```

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.

```

3962     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3963     {
3964         { \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`.

```

3965         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3966         { \int_use:N \l_@@_final_i_int }
3967         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3968         { } % for the name of the block
3969     }
3970 }

```

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).

```

3971 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3972 {
3973     \int_set:Nn \l_@@_row_min_int 1
3974     \int_set:Nn \l_@@_col_min_int 1
3975     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3976     \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`.

```

3977     \seq_map_inline:Nn \g_@@_submatrix_seq
3978     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
3979 }

```

#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.

```

3980 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3981 {
3982   \bool_if:nT
3983   {
3984     \int_compare_p:n { #3 <= #1 }
3985     && \int_compare_p:n { #1 <= #5 }
3986     && \int_compare_p:n { #4 <= #2 }
3987     && \int_compare_p:n { #2 <= #6 }
3988   }
3989   {
3990     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3991     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3992     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3993     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3994   }
3995 }

3996 \cs_new_protected:Npn \@@_set_initial_coords:
3997 {
3998   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3999   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4000 }
4001 \cs_new_protected:Npn \@@_set_final_coords:
4002 {
4003   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4004   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4005 }
4006 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4007 {
4008   \pgfpointanchor
4009   {
4010     \@@_env:
4011     - \int_use:N \l_@@_initial_i_int
4012     - \int_use:N \l_@@_initial_j_int
4013   }
4014   { #1 }
4015   \@@_set_initial_coords:
4016 }
4017 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4018 {
4019   \pgfpointanchor
4020   {
4021     \@@_env:
4022     - \int_use:N \l_@@_final_i_int
4023     - \int_use:N \l_@@_final_j_int
4024   }
4025   { #1 }
4026   \@@_set_final_coords:
4027 }
4028 \cs_new_protected:Npn \@@_open_x_initial_dim:
4029 {
4030   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4031   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4032   {
4033     \cs_if_exist:cT
4034     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4035     {
4036       \pgfpointanchor
4037       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4038       { west }

```

```

4039         \dim_set:Nn \l_@@_x_initial_dim
4040         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4041     }
4042 }

```

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

```

4043 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4044 {
4045     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4046     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4047     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4048 }
4049 }
4050 \cs_new_protected:Npn \@@_open_x_final_dim:
4051 {
4052     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4053     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4054     {
4055         \cs_if_exist:cT
4056         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4057         {
4058             \pgfpointanchor
4059             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4060             { east }
4061             \dim_set:Nn \l_@@_x_final_dim
4062             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4063         }
4064     }

```

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

```

4065 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4066 {
4067     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4068     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4069     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4070 }
4071 }

```

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.

```

4072 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4073 {
4074     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4075     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4076     {
4077         \@@_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.

```

4078 \group_begin:
4079     \int_compare:nNnTF { #1 } = 0
4080     { \color { nicematrix-first-row } }
4081     {

```

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.

```

4082         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4083         { \color { nicematrix-last-row } }
4084     }
4085     \keys_set:nn { NiceMatrix / xdots } { #3 }
4086     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4087     \@@_actually_draw_Ldots:
4088 \group_end:

```

```

4089     }
4090 }

```

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`.

```

4091 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4092 {
4093   \bool_if:NTF \l_@@_initial_open_bool
4094   {
4095     \@@_open_x_initial_dim:
4096     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4097     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4098   }
4099   { \@@_set_initial_coords_from_anchor:n { base-east } }
4100   \bool_if:NTF \l_@@_final_open_bool
4101   {
4102     \@@_open_x_final_dim:
4103     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4104     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4105   }
4106   { \@@_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 (?).

```

4107   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4108   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4109   \@@_draw_line:
4110 }

```

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.

```

4111 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4112 {
4113   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4114   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4115   {
4116     \@@_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.

```

4117   \group_begin:
4118     \int_compare:nNnTF { #1 } = 0
4119     { \color { nicematrix-first-row } }
4120     {

```

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.

```

4121     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4122     { \color { nicematrix-last-row } }
4123   }
4124   \keys_set:nn { NiceMatrix / xdots } { #3 }

```

```

4125         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4126         \@@_actually_draw_Cdots:
4127     \group_end:
4128 }
4129 }

```

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.`

```

4130 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4131 {
4132     \bool_if:NTF \l_@@_initial_open_bool
4133     { \@@_open_x_initial_dim: }
4134     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4135     \bool_if:NTF \l_@@_final_open_bool
4136     { \@@_open_x_final_dim: }
4137     { \@@_set_final_coords_from_anchor:n { mid-west } }
4138     \bool_lazy_and:nnTF
4139     \l_@@_initial_open_bool
4140     \l_@@_final_open_bool
4141     {
4142         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4143         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4144         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4145         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4146         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4147     }
4148     {
4149         \bool_if:NT \l_@@_initial_open_bool
4150         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4151         \bool_if:NT \l_@@_final_open_bool
4152         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4153     }
4154     \@@_draw_line:
4155 }
4156 \cs_new_protected:Npn \@@_open_y_initial_dim:
4157 {
4158     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4159     \dim_set:Nn \l_@@_y_initial_dim
4160     {
4161         \fp_to_dim:n
4162         {
4163             \pgf@y
4164             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4165         }
4166     } % modified 6.13c
4167     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4168     {
4169         \cs_if_exist:cT
4170         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4171         {
4172             \pgfpointanchor
4173             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4174             { north }

```

```

4175         \dim_set:Nn \l_@@_y_initial_dim
4176         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4177     }
4178 }
4179 }
4180 \cs_new_protected:Npn \@@_open_y_final_dim:
4181 {
4182     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4183     \dim_set:Nn \l_@@_y_final_dim
4184     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4185     % modified 6.13c
4186     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4187     {
4188         \cs_if_exist:cT
4189         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4190         {
4191             \pgfpointanchor
4192             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4193             { south }
4194             \dim_set:Nn \l_@@_y_final_dim
4195             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4196         }
4197     }
4198 }

```

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.

```

4199 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4200 {
4201     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4202     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4203     {
4204         \@@_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.

```

4205     \group_begin:
4206     \int_compare:nNnTF { #2 } = 0
4207     { \color { nicematrix-first-col } }
4208     {
4209         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4210         { \color { nicematrix-last-col } }
4211     }
4212     \keys_set:nn { NiceMatrix / xdots } { #3 }
4213     \tl_if_empty:VF \l_@@_xdots_color_tl
4214     { \color { \l_@@_xdots_color_tl } }
4215     \@@_actually_draw_Vdots:
4216     \group_end:
4217 }
4218 }

```

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`.

```
4219 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4220 {
```

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

```
4221 \bool_set_false:N \l_tmpa_bool
```

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

```
4222 \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
4223 {
4224   \@@_set_initial_coords_from_anchor:n { south-west }
4225   \@@_set_final_coords_from_anchor:n { north-west }
4226   \bool_set:Nn \l_tmpa_bool
4227   { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4228 }
```

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

```
4229 \bool_if:NTF \l_@@_initial_open_bool
4230 \@@_open_y_initial_dim:
4231 { \@@_set_initial_coords_from_anchor:n { south } }
4232 \bool_if:NTF \l_@@_final_open_bool
4233 \@@_open_y_final_dim:
4234 { \@@_set_final_coords_from_anchor:n { north } }
4235 \bool_if:NTF \l_@@_initial_open_bool
4236 {
4237   \bool_if:NTF \l_@@_final_open_bool
4238   {
4239     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4240     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4241     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4242     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4243     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4244   }
4245 }
```

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.

```
4244 \int_compare:nNnT \l_@@_last_col_int > { -2 }
4245 {
4246   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
4247   {
4248     \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
4249     \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
4250     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4251     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
4252   }
4253 }
4254 }
4255 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
4256 }
4257 {
4258   \bool_if:NTF \l_@@_final_open_bool
4259   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
4260 }
```

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.

```
4261 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4262 {
4263   \dim_set:Nn \l_@@_x_initial_dim
4264   {
4265     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4266     \l_@@_x_initial_dim \l_@@_x_final_dim
4267   }
4268   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4269 }
```

```

4270     }
4271   }
4272   \@@_draw_line:
4273 }

```

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.

```

4274 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4275 {
4276   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4277   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4278   {
4279     \@@_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.

```

4280     \group_begin:
4281       \keys_set:nn { NiceMatrix / xdots } { #3 }
4282       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4283       \@@_actually_draw_Ddots:
4284     \group_end:
4285   }
4286 }

```

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`.

```

4287 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4288 {
4289   \bool_if:NTF \l_@@_initial_open_bool
4290   {
4291     \@@_open_y_initial_dim:
4292     \@@_open_x_initial_dim:
4293   }
4294   { \@@_set_initial_coords_from_anchor:n { south-east } }
4295   \bool_if:NTF \l_@@_final_open_bool
4296   {
4297     \@@_open_x_final_dim:
4298     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4299   }
4300   { \@@_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.

```

4301   \bool_if:NT \l_@@_parallelize_diags_bool
4302   {
4303     \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).

```

4304     \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.

```

4305     {
4306         \dim_gset:Nn \g_@@_delta_x_one_dim
4307         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4308         \dim_gset:Nn \g_@@_delta_y_one_dim
4309         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4310     }

```

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`.

```

4311     {
4312         \dim_set:Nn \l_@@_y_final_dim
4313         {
4314             \l_@@_y_initial_dim +
4315             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4316             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4317         }
4318     }
4319 }
4320 \@@_draw_line:
4321 }

```

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.

```

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

```

4328     \group_begin:
4329     \keys_set:nn { NiceMatrix / xdots } { #3 }
4330     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4331     \@@_actually_draw_Iddots:
4332     \group_end:
4333 }
4334 }

```

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`.

```

4335 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4336 {
4337     \bool_if:NTF \l_@@_initial_open_bool
4338     {
4339         \@@_open_y_initial_dim:
4340         \@@_open_x_initial_dim:

```

```

4341     }
4342     { \@@_set_initial_coords_from_anchor:n { south-west } }
4343     \bool_if:NTF \l_@@_final_open_bool
4344     {
4345         \@@_open_y_final_dim:
4346         \@@_open_x_final_dim:
4347     }
4348     { \@@_set_final_coords_from_anchor:n { north-east } }
4349     \bool_if:NT \l_@@_parallelize_diags_bool
4350     {
4351         \int_gincr:N \g_@@_iddots_int
4352         \int_compare:nNnTF \g_@@_iddots_int = 1
4353         {
4354             \dim_gset:Nn \g_@@_delta_x_two_dim
4355             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4356             \dim_gset:Nn \g_@@_delta_y_two_dim
4357             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4358         }
4359         {
4360             \dim_set:Nn \l_@@_y_final_dim
4361             {
4362                 \l_@@_y_initial_dim +
4363                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4364                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4365             }
4366         }
4367     }
4368     \@@_draw_line:
4369 }

```

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`

```

4370 \cs_new_protected:Npn \@@_draw_line:
4371 {
4372     \pgfrememberpicturepositiononpagetrue
4373     \pgf@relevantforpicturesizefalse
4374     \bool_lazy_or:nnTF
4375     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4376     \l_@@_dotted_bool
4377     \@@_draw_standard_dotted_line:
4378     \@@_draw_unstandard_dotted_line:
4379 }

```

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.

```

4380 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4381 {
4382   \begin { scope }
4383   \@@_draw_unstandard_dotted_line:o
4384   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4385 }

```

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.

```

4386 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4387 {
4388   \@@_draw_unstandard_dotted_line:nVV
4389   { #1 }
4390   \l_@@_xdots_up_tl
4391   \l_@@_xdots_down_tl
4392 }
4393 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4394 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
4395 {
4396   \draw
4397   [
4398     #1 ,
4399     shorten-> = \l_@@_xdots_shorten_end_dim ,
4400     shorten-< = \l_@@_xdots_shorten_start_dim ,
4401   ]
4402   ( \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).

```

4403   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
4404   node [ sloped , below ] { $ \scriptstyle #3 $ }
4405   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4406   \end { scope }
4407 }
4408 \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).

```

4409 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4410 {
4411   \bool_lazy_and:nnF
4412   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4413   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4414   {
4415     \pgfscope
4416     \pgftransformshift
4417     {
4418       \pgfpointlineattime { 0.5 }
4419       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4420       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4421     }
4422     \pgftransformrotate
4423     {
4424       \fp_eval:n
4425       {
4426         atand
4427         (
4428           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4429           \l_@@_x_final_dim - \l_@@_x_initial_dim

```

```

4430         )
4431     }
4432 }
4433 \pgfnode
4434 { rectangle }
4435 { south }
4436 {
4437     \c_math_toggle_token
4438     \scriptstyle \l_@@_xdots_up_tl
4439     \c_math_toggle_token
4440 }
4441 { }
4442 { \pgfusepath { } }
4443 \pgfnode
4444 { rectangle }
4445 { north }
4446 {
4447     \c_math_toggle_token
4448     \scriptstyle \l_@@_xdots_down_tl
4449     \c_math_toggle_token
4450 }
4451 { }
4452 { \pgfusepath { } }
4453 \endpgfscope
4454 }
4455 \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.

```

4456 \dim_zero_new:N \l_@@_l_dim
4457 \dim_set:Nn \l_@@_l_dim
4458 {
4459     \fp_to_dim:n
4460     {
4461         sqrt
4462         (
4463             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4464             +
4465             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4466         )
4467     }
4468 }

```

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.

```

4469 \bool_lazy_or:nnF
4470 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4471 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4472 \@@_draw_standard_dotted_line_i:
4473 \group_end:
4474 }
4475 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4476 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4477 {

```

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

```

4478 \bool_if:NTF \l_@@_initial_open_bool
4479 {
4480     \bool_if:NTF \l_@@_final_open_bool
4481     {
4482         \int_set:Nn \l_tmpa_int

```

```

4483         { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
4484     }
4485     {
4486         \int_set:Nn \l_tmpa_int
4487         {
4488             \dim_ratio:nn
4489             { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
4490             \l_@@_xdots_inter_dim
4491         }
4492     }
4493 }
4494 {
4495     \bool_if:NTF \l_@@_final_open_bool
4496     {
4497         \int_set:Nn \l_tmpa_int
4498         {
4499             \dim_ratio:nn
4500             { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
4501             \l_@@_xdots_inter_dim
4502         }
4503     }
4504     {
4505         \int_set:Nn \l_tmpa_int
4506         {
4507             \dim_ratio:nn
4508             {
4509                 \l_@@_l_dim
4510                 - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4511             }
4512             \l_@@_xdots_inter_dim
4513         }
4514     }
4515 }

```

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

```

4516     \dim_set:Nn \l_tmpa_dim
4517     {
4518         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4519         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4520     }
4521     \dim_set:Nn \l_tmpb_dim
4522     {
4523         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4524         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4525     }

```

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.

```

4526     \dim_gadd:Nn \l_@@_x_initial_dim
4527     {
4528         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4529         \dim_ratio:nn
4530         {
4531             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4532             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4533         }
4534         { 2 \l_@@_l_dim }
4535     }
4536     \dim_gadd:Nn \l_@@_y_initial_dim
4537     {
4538         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4539         \dim_ratio:nn
4540         {

```

```

4541         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4542         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4543     }
4544     { 2 \l_@@_l_dim }
4545 }
4546 \pgf@relevantforpicturesizefalse
4547 \int_step_inline:nnn 0 \l_tmpa_int
4548 {
4549     \pgfpathcircle
4550     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4551     { \l_@@_xdots_radius_dim }
4552     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4553     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4554 }
4555 \pgfusepathqfill
4556 }

```

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.

```

4557 \hook_gput_code:nnn { begindocument } { . }
4558 {
4559     \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
4560     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4561     \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4562     {
4563         \int_compare:nNnTF \c@jCol = 0
4564         { \@@_error:nn { in~first~col } \Ldots }
4565         {
4566             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4567             { \@@_error:nn { in~last~col } \Ldots }
4568             {
4569                 \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
4570                 { #1 , down = #2 , up = #3 }
4571             }
4572         }
4573         \bool_if:NF \l_@@_nullify_dots_bool
4574         { \phantom { \ensuremath { \@@_old_ldots } } }
4575         \bool_gset_true:N \g_@@_empty_cell_bool
4576     }

4577 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4578 {
4579     \int_compare:nNnTF \c@jCol = 0
4580     { \@@_error:nn { in~first~col } \Cdots }
4581     {
4582         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4583         { \@@_error:nn { in~last~col } \Cdots }
4584         {
4585             \@@_instruction_of_type:nnn \c_false_bool { \Cdots }

```

```

4586         { #1 , down = #2 , up = #3 }
4587     }
4588 }
4589 \bool_if:NF \l_@@_nullify_dots_bool
4590 { \phantom { \ensuremath { \@@_old_cdots } } }
4591 \bool_gset_true:N \g_@@_empty_cell_bool
4592 }

4593 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4594 {
4595     \int_compare:nNnTF \c@iRow = 0
4596     { \@@_error:nn { in~first~row } \Vdots }
4597     {
4598         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4599         { \@@_error:nn { in~last~row } \Vdots }
4600         {
4601             \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4602             { #1 , down = #2 , up = #3 }
4603         }
4604     }
4605     \bool_if:NF \l_@@_nullify_dots_bool
4606     { \phantom { \ensuremath { \@@_old_vdots } } }
4607     \bool_gset_true:N \g_@@_empty_cell_bool
4608 }

4609 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4610 {
4611     \int_case:nnF \c@iRow
4612     {
4613         0 { \@@_error:nn { in~first~row } \Ddots }
4614         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4615     }
4616     {
4617         \int_case:nnF \c@jCol
4618         {
4619             0 { \@@_error:nn { in~first~col } \Ddots }
4620             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4621         }
4622         {
4623             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4624             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
4625             { #1 , down = #2 , up = #3 }
4626         }
4627     }
4628 }
4629 \bool_if:NF \l_@@_nullify_dots_bool
4630 { \phantom { \ensuremath { \@@_old_ddots } } }
4631 \bool_gset_true:N \g_@@_empty_cell_bool
4632 }

4633 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4634 {
4635     \int_case:nnF \c@iRow
4636     {
4637         0 { \@@_error:nn { in~first~row } \Iddots }
4638         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4639     }
4640     {
4641         \int_case:nnF \c@jCol
4642         {
4643             0 { \@@_error:nn { in~first~col } \Iddots }

```

```

4644         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4645     }
4646     {
4647         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4648         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4649         { #1 , down = #2 , up = #3 }
4650     }
4651 }
4652 \bool_if:NF \l_@@_nullify_dots_bool
4653 { \phantom { \ensuremath { \@@_old_iddots } } }
4654 \bool_gset_true:N \g_@@_empty_cell_bool
4655 }
4656 }

```

End of the \AddToHook.

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

```

4657 \keys_define:nn { NiceMatrix / Ddots }
4658 {
4659     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4660     draw-first .default:n = true ,
4661     draw-first .value_forbidden:n = true
4662 }

```

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

```

4663 \cs_new_protected:Npn \@@_Hspace:
4664 {
4665     \bool_gset_true:N \g_@@_empty_cell_bool
4666     \hspace
4667 }

```

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.

```

4668 \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.

```

4669 \cs_new:Npn \@@_Hdotsfor:
4670 {
4671     \bool_lazy_and:nnTF
4672     { \int_compare_p:nNn \c@jCol = 0 }
4673     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4674     {
4675         \bool_if:NTF \g_@@_after_col_zero_bool
4676         {
4677             \multicolumn { 1 } { c } { }
4678             \@@_Hdotsfor_i
4679         }
4680         { \@@_fatal:n { Hdotsfor~in~col~0 } }
4681     }
4682     {
4683         \multicolumn { 1 } { c } { }
4684         \@@_Hdotsfor_i
4685     }
4686 }

```

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:).

```

4687 \hook_gput_code:nnn { begindocument } { . }

```



```

4688 {
4689   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4690   \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.

```

4691   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4692   {
4693     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4694     {
4695       \@@_Hdotsfor:nnnn
4696       { \int_use:N \c@iRow }
4697       { \int_use:N \c@jCol }
4698       { #2 }
4699       {
4700         #1 , #3 ,
4701         down = \exp_not:n { #4 } ,
4702         up = \exp_not:n { #5 }
4703       }
4704     }
4705     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4706   }
4707 }

```

```

4708 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4709 {
4710   \bool_set_false:N \l_@@_initial_open_bool
4711   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4712   \int_set:Nn \l_@@_initial_i_int { #1 }
4713   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4714   \int_compare:nNnTF { #2 } = 1
4715   {
4716     \int_set:Nn \l_@@_initial_j_int 1
4717     \bool_set_true:N \l_@@_initial_open_bool
4718   }
4719   {
4720     \cs_if_exist:cTF
4721     {
4722       pgf @ sh @ ns @ \@@_env:
4723       - \int_use:N \l_@@_initial_i_int
4724       - \int_eval:n { #2 - 1 }
4725     }
4726     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4727     {
4728       \int_set:Nn \l_@@_initial_j_int { #2 }
4729       \bool_set_true:N \l_@@_initial_open_bool
4730     }
4731   }
4732   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4733   {
4734     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4735     \bool_set_true:N \l_@@_final_open_bool
4736   }
4737   {
4738     \cs_if_exist:cTF
4739     {
4740       pgf @ sh @ ns @ \@@_env:
4741       - \int_use:N \l_@@_final_i_int
4742       - \int_eval:n { #2 + #3 }
4743     }

```

```

4744     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4745     {
4746         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4747         \bool_set_true:N \l_@@_final_open_bool
4748     }
4749 }
4750 \group_begin:
4751 \int_compare:nNnTF { #1 } = 0
4752 { \color { nicematrix-first-row } }
4753 {
4754     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4755     { \color { nicematrix-last-row } }
4756 }
4757 \keys_set:nn { NiceMatrix / xdots } { #4 }
4758 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4759 \@@_actually_draw_Ldots:
4760 \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).

```

4761     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4762     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4763 }

4764 \hook_gput_code:nnn { begindocument } { . }
4765 {
4766     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4767     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4768     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4769     {
4770         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4771         {
4772             \@@_Vdotsfor:nnnn
4773             { \int_use:N \c@iRow }
4774             { \int_use:N \c@jCol }
4775             { #2 }
4776             {
4777                 #1 , #3 ,
4778                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4779             }
4780         }
4781     }
4782 }

```

Enf of `\AddToHook`.

```

4783 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4784 {
4785     \bool_set_false:N \l_@@_initial_open_bool
4786     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

4787     \int_set:Nn \l_@@_initial_j_int { #2 }
4788     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4789     \int_compare:nNnTF #1 = 1
4790     {
4791         \int_set:Nn \l_@@_initial_i_int 1
4792         \bool_set_true:N \l_@@_initial_open_bool
4793     }
4794     {
4795         \cs_if_exist:CTF

```

```

4796     {
4797         pgf @ sh @ ns @ \@@_env:
4798         - \int_eval:n { #1 - 1 }
4799         - \int_use:N \l_@@_initial_j_int
4800     }
4801     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4802     {
4803         \int_set:Nn \l_@@_initial_i_int { #1 }
4804         \bool_set_true:N \l_@@_initial_open_bool
4805     }
4806 }
4807 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4808 {
4809     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4810     \bool_set_true:N \l_@@_final_open_bool
4811 }
4812 {
4813     \cs_if_exist:cTF
4814     {
4815         pgf @ sh @ ns @ \@@_env:
4816         - \int_eval:n { #1 + #3 }
4817         - \int_use:N \l_@@_final_j_int
4818     }
4819     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4820     {
4821         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4822         \bool_set_true:N \l_@@_final_open_bool
4823     }
4824 }
4825 \group_begin:
4826 \int_compare:nNnTF { #2 } = 0
4827 { \color { nicematrix-first-col } }
4828 {
4829     \int_compare:nNnT { #2 } = \g_@@_col_total_int
4830     { \color { nicematrix-last-col } }
4831 }
4832 \keys_set:nn { NiceMatrix / xdots } { #4 }
4833 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4834 \@@_actually_draw_Vdots:
4835 \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).

```

4836     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4837     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4838 }

```

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

```

4839 \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.

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

```

4840 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4841 {
4842   \tl_if_empty:nTF { #2 }
4843     { #1 }
4844     { \@@_double_int_eval_i:n #1-#2 \q_stop }
4845 }
4846 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
4847 { \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).

```

4848 \hook_gput_code:nnn { begindocument } { . }
4849 {
4850   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4851   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4852   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4853     {
4854       \group_begin:
4855       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4856       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4857       \use:e
4858       {
4859         \@@_line_i:nn
4860         { \@@_double_int_eval:n #2 - \q_stop }
4861         { \@@_double_int_eval:n #3 - \q_stop }
4862       }
4863       \group_end:
4864     }
4865 }
4866 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4867 {
4868   \bool_set_false:N \l_@@_initial_open_bool
4869   \bool_set_false:N \l_@@_final_open_bool
4870   \bool_if:nTF
4871     {
4872       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4873       ||
4874       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4875     }
4876     {
4877       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4878     }
4879     { \@@_draw_line_ii:nn { #1 } { #2 } }
4880 }
4881 \hook_gput_code:nnn { begindocument } { . }
4882 {
4883   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4884   {

```

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

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible” and that why we do this static construction of the command `\@@_draw_line_ii:`.

```

4885     \c_@@_pgfortikzpicture_tl
4886     \@@_draw_line_iii:nn { #1 } { #2 }
4887     \c_@@_endpgfortikzpicture_tl
4888   }
4889 }

```

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

```

4890 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4891 {
4892   \pgfrememberpicturepositiononpagetrue
4893   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4894   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4895   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4896   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4897   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4898   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4899   \@@_draw_line:
4900 }

```

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`

```

4901 \keys_define:nn { NiceMatrix / RowStyle }
4902 {
4903   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4904   cell-space-top-limit .initial:n = \c_zero_dim ,
4905   cell-space-top-limit .value_required:n = true ,
4906   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4907   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4908   cell-space-bottom-limit .value_required:n = true ,
4909   cell-space-limits .meta:n =
4910   {
4911     cell-space-top-limit = #1 ,
4912     cell-space-bottom-limit = #1 ,
4913   } ,
4914   color .tl_set:N = \l_@@_color_tl ,
4915   color .value_required:n = true ,
4916   bold .bool_set:N = \l_tmpa_bool ,
4917   bold .default:n = true ,
4918   bold .initial:n = false ,
4919   nb-rows .code:n =
4920   \str_if_eq:nnTF { #1 } { * }
4921   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
4922   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
4923   nb-rows .value_required:n = true ,
4924   rowcolor .tl_set:N = \l_tmpa_tl ,
4925   rowcolor .value_required:n = true ,
4926   rowcolor .initial:n = ,
4927   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
4928 }

4929 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4930 {

```

```

4931 \group_begin:
4932 \tl_clear:N \l_tmpa_tl % value of \rowcolor
4933 \tl_clear:N \l_@@_color_tl
4934 \int_set:Nn \l_@@_key_nb_rows_int 1
4935 \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4936 \tl_if_empty:NF \l_tmpa_tl
4937 {

```

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

```

4938 \tl_gput_right:Nx \g_@@_pre_code_before_tl
4939 {

```

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

```

4940 \@@_exp_color_arg:NV \@@_rectanglecolor \l_tmpa_tl
4941 { \int_use:N \c@iRow - \int_use:N \c@jCol }
4942 { \int_use:N \c@iRow - * }
4943 }

```

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

```

4944 \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4945 {
4946   \tl_gput_right:Nx \g_@@_pre_code_before_tl
4947   {
4948     \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
4949     {
4950       \int_eval:n { \c@iRow + 1 }
4951       - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4952     }
4953   }
4954 }
4955 }
4956 \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4957 \tl_gput_right:Nx \g_@@_row_style_tl
4958 { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4959 \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

4960 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4961 {
4962   \tl_gput_right:Nx \g_@@_row_style_tl
4963   {
4964     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4965     {
4966       \dim_set:Nn \l_@@_cell_space_top_limit_dim
4967       { \dim_use:N \l_tmpa_dim }
4968     }
4969   }
4970 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

4971 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4972 {
4973   \tl_gput_right:Nx \g_@@_row_style_tl
4974   {
4975     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4976     {
4977       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4978       { \dim_use:N \l_tmpb_dim }
4979     }
4980   }
4981 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

4982 \tl_if_empty:NF \l_@@_color_tl
4983 {
4984   \tl_gput_right:Nx \g_@@_row_style_tl

```

```

4985         {
4986             \mode_leave_vertical:
4987             \@@_color:n { \l_@@_color_tl }
4988         }
4989     }
\l_tmpa_bool is the value of the key bold.
4990     \bool_if:NT \l_tmpa_bool
4991     {
4992         \tl_gput_right:Nn \g_@@_row_style_tl
4993         {
4994             \if_mode_math:
4995                 \c_math_toggle_token
4996                 \bfseries \boldmath
4997                 \c_math_toggle_token
4998             \else:
4999                 \bfseries \boldmath
5000             \fi:
5001         }
5002     }
5003     \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5004     \group_end:
5005     \g_@@_row_style_tl
5006     \ignorespaces
5007 }

```

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).

```

5008 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5009 {

```

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.

```

5010     \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`.

```

5011     \str_if_in:nnF { #1 } { !! }
5012     {
5013         \seq_map_indexed_inline:Nn \g_@@_colors_seq
5014         { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5015     }
5016     \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

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

```

5017 {
5018   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5019   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5020 }

```

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

```

5021 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5022 }
5023 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5024 \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`).

```

5025 \cs_new_protected:Npn \@@_actually_color:
5026 {
5027   \pgfpicture
5028   \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.

```

5029   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5030   {
5031     \pgfsetcornersarced
5032     {
5033       \pgfpoint
5034       { \l_@@_tab_rounded_corners_dim }
5035       { \l_@@_tab_rounded_corners_dim }
5036     }
5037   %   \end{macrocode}
5038   % Because we want \pkg{nicematrix} compatible with arrays constructed by
5039   % \pkg{array}, the nodes for the rows and columns (that is to say the nodes
5040   % |row-|\textsl{i} and |col-|\textsl{j}|) have not always the expected position,
5041   % that is to say, there is sometimes a slight shifting of something such as
5042   % |\arrayrulewidth|. Now, for the clipping, we have to change slightly the
5043   % position of that clipping whether a rounded rectangle around the array is
5044   % required. That's the point which is tested in the following line.
5045   %   \begin{macrocode}
5046     \bool_if:NTF \l_@@_hvlines_bool
5047     {
5048       \pgfpathrectanglecorners
5049       {
5050         \pgfpointadd
5051         { \@@_qpoint:n { row-1 } }
5052         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5053       }
5054       {
5055         \pgfpointadd
5056         {
5057           \@@_qpoint:n
5058           { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5059         }
5060         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5061       }
5062     }
5063     {
5064       \pgfpathrectanglecorners
5065       { \@@_qpoint:n { row-1 } }
5066       {
5067         \pgfpointadd
5068         {
5069           \@@_qpoint:n

```



```

5070          { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5071        }
5072        { \pgfpicture \c_zero_dim \arrayrulewidth }
5073      }
5074    }
5075    \pgfusepath { clip }
5076  }
5077  \seq_map_indexed_inline:Nn \g_@@_colors_seq
5078  {
5079    \begin { pgfscope }
5080      \@@_color_opacity ##2
5081      \use:c { g_@@_color _ ##1 _tl }
5082      \tl_gclear:c { g_@@_color _ ##1 _tl }
5083      \pgfusepath { fill }
5084    \end { pgfscope }
5085  }
5086  \endpgfpicture
5087 }

```

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

```

5088 \cs_new_protected:Npn \@@_color_opacity
5089 {
5090   \peek_meaning:NTF [
5091     { \@@_color_opacity:w }
5092     { \@@_color_opacity:w [ ] }
5093   }

```

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

```

5094 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5095 {
5096   \tl_clear:N \l_tmpa_tl
5097   \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.

```

5098   \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5099   \tl_if_empty:NTF \l_tmpb_tl
5100     { \@declaredcolor }
5101     { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5102 }

```

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

```

5103 \keys_define:nn { nicematrix / color-opacity }
5104 {
5105   opacity .tl_set:N          = \l_tmpa_tl ,
5106   opacity .value_required:n = true
5107 }

5108 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5109 {
5110   \tl_set:Nn \l_@@_rows_tl { #1 }
5111   \tl_set:Nn \l_@@_cols_tl { #2 }
5112   \@@_cartesian_path:
5113 }

```

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

```

5114 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5115 {

```

```

5116 \tl_if_blank:nF { #2 }
5117 {
5118   \@@_add_to_colors_seq:xn
5119   { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5120   { \@@_cartesian_color:nn { #3 } { - } }
5121 }
5122 }

```

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

```

5123 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5124 {
5125   \tl_if_blank:nF { #2 }
5126   {
5127     \@@_add_to_colors_seq:xn
5128     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5129     { \@@_cartesian_color:nn { - } { #3 } }
5130   }
5131 }

```

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

```

5132 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5133 {
5134   \tl_if_blank:nF { #2 }
5135   {
5136     \@@_add_to_colors_seq:xn
5137     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5138     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5139   }
5140 }

```

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

```

5141 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5142 {
5143   \tl_if_blank:nF { #2 }
5144   {
5145     \@@_add_to_colors_seq:xn
5146     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5147     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5148   }
5149 }

```

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

```

5150 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5151 {
5152   \@@_cut_on_hyphen:w #1 \q_stop
5153   \tl_clear_new:N \l_@@_tmpc_tl
5154   \tl_clear_new:N \l_@@_tmpd_tl
5155   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5156   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5157   \@@_cut_on_hyphen:w #2 \q_stop
5158   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5159   \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`.

```

5160   \@@_cartesian_path:n { #3 }
5161 }

```

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

```

5162 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5163 {

```

```

5164 \clist_map_inline:nn { #3 }
5165 { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5166 }

5167 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5168 {
5169   \int_step_inline:nn { \int_use:N \c@iRow }
5170   {
5171     \int_step_inline:nn { \int_use:N \c@jCol }
5172     {
5173       \int_if_even:nTF { ####1 + ##1 }
5174       { \@@_cellcolor [ #1 ] { #2 } }
5175       { \@@_cellcolor [ #1 ] { #3 } }
5176       { ##1 - ####1 }
5177     }
5178   }
5179 }

```

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”.

```

5180 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5181 {
5182   \@@_rectanglecolor [ #1 ] { #2 }
5183   { 1 - 1 }
5184   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5185 }

5186 \keys_define:nn { NiceMatrix / rowcolors }
5187 {
5188   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5189   respect-blocks .default:n = true ,
5190   cols .tl_set:N = \l_@@_cols_tl ,
5191   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5192   restart .default:n = true ,
5193   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5194 }

```

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*.

```

5195 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5196 {

```

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

```

5197   \group_begin:
5198   \seq_clear_new:N \l_@@_colors_seq
5199   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5200   \tl_clear_new:N \l_@@_cols_tl
5201   \tl_set:Nn \l_@@_cols_tl { - }
5202   \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).

```

5203   \int_zero_new:N \l_@@_color_int
5204   \int_set:Nn \l_@@_color_int 1
5205   \bool_if:NT \l_@@_respect_blocks_bool
5206   {

```

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`).

```

5207     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5208     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5209     { \@@_not_in_exterior_p:nnnnn ##1 }
5210   }
5211   \pgfpicture
5212   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5213   \clist_map_inline:nn { #2 }
5214   {
5215     \tl_set:Nn \l_tmpa_tl { ##1 }
5216     \tl_if_in:NnTF \l_tmpa_tl { - }
5217     { \@@_cut_on_hyphen:w ##1 \q_stop }
5218     { \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.

```

5219     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5220     \bool_if:NTF \l_@@_rowcolors_restart_bool
5221     { \int_set:Nn \l_@@_color_int 1 }
5222     { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
5223     \int_zero_new:N \l_@@_tmpc_int
5224     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5225     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5226     {

```

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

```

5227         \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).

```

5228         \bool_if:NT \l_@@_respect_blocks_bool
5229         {
5230           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5231           { \@@_intersect_our_row_p:nnnnn #####1 }
5232           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5233       }
5234       \tl_set:Nx \l_@@_rows_tl
5235       { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5236       \tl_clear_new:N \l_@@_color_tl
5237       \tl_set:Nx \l_@@_color_tl
5238       {
5239         \@@_color_index:n
5240         {
5241           \int_mod:nn
5242           { \l_@@_color_int - 1 }
5243           { \seq_count:N \l_@@_colors_seq }
5244           + 1
5245         }
5246       }
5247       \tl_if_empty:NF \l_@@_color_tl
5248       {
5249         \@@_add_to_colors_seq:xx
5250         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5251         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5252       }
5253       \int_incr:N \l_@@_color_int
5254       \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5255     }

```

```

5256     }
5257     \endpgfpicture
5258     \group_end:
5259 }

```

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.

```

5260 \cs_new:Npn \@@_color_index:n #1
5261 {
5262     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5263     { \@@_color_index:n { #1 - 1 } }
5264     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5265 }

```

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

```

5266 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
5267 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

5268 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5269 {
5270     \int_compare:nNnT { #3 } > \l_tmpb_int
5271     { \int_set:Nn \l_tmpb_int { #3 } }
5272 }

5273 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5274 {
5275     \bool_lazy_or:nnTF
5276     { \int_compare_p:nNn { #4 } = \c_zero_int }
5277     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c_jCol + 1 } } }
5278     \prg_return_false:
5279     \prg_return_true:
5280 }

```

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

```

5281 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5282 {
5283     \bool_if:nTF
5284     {
5285         \int_compare_p:n { #1 <= \l_tmpa_int }
5286         &&
5287         \int_compare_p:n { \l_tmpa_int <= #3 }
5288     }
5289     \prg_return_true:
5290     \prg_return_false:
5291 }

```

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`).

```

5292 \cs_new_protected:Npn \@@_cartesian_path:n #1
5293 {
5294     \bool_lazy_and:nnT
5295     { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5296     { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5297     {

```

```

5298 \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5299 \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5300 }

```

We begin the loop over the columns.

```

5301 \clist_map_inline:Nn \l_@@_cols_tl
5302 {
5303   \tl_set:Nn \l_tmpa_tl { ##1 }
5304   \tl_if_in:NnTF \l_tmpa_tl { - }
5305     { \@@_cut_on_hyphen:w ##1 \q_stop }
5306     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5307   \bool_lazy_or:nnT
5308     { \tl_if_blank_p:V \l_tmpa_tl }
5309     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5310     { \tl_set:Nn \l_tmpa_tl { 1 } }
5311   \bool_lazy_or:nnT
5312     { \tl_if_blank_p:V \l_tmpb_tl }
5313     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5314     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5315   \int_compare:nNnT \l_tmpb_tl > \c@jCol
5316     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

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

```

5317 \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.

```

5318 \@@_qpoint:n { col - \l_tmpa_tl }
5319 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5320   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5321   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5322 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5323 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5324 \clist_map_inline:Nn \l_@@_rows_tl
5325 {
5326   \tl_set:Nn \l_tmpa_tl { #####1 }
5327   \tl_if_in:NnTF \l_tmpa_tl { - }
5328     { \@@_cut_on_hyphen:w #####1 \q_stop }
5329     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5330   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5331   \tl_if_empty:NT \l_tmpb_tl
5332     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5333   \int_compare:nNnT \l_tmpb_tl > \c@iRow
5334     { \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`.

```

5335 \seq_if_in:NxF \l_@@_corners_cells_seq
5336   { \l_tmpa_tl - \l_@@_tmpc_tl }
5337   {
5338     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5339     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5340     \@@_qpoint:n { row - \l_tmpa_tl }
5341     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5342     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5343     \pgfpathrectanglecorners
5344       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5345       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5346   }
5347 }
5348 }
5349 }

```

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`).

```
5350 \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.

```
5351 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5352 {
5353   \clist_set_eq:NN \l_tmpa_clist #1
5354   \clist_clear:N #1
5355   \clist_map_inline:Nn \l_tmpa_clist
5356   {
5357     \tl_set:Nn \l_tmpa_tl { ##1 }
5358     \tl_if_in:NnTF \l_tmpa_tl { - }
5359     { \@@_cut_on_hyphen:w ##1 \q_stop }
5360     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5361     \bool_lazy_or:nnT
5362     { \tl_if_blank_p:V \l_tmpa_tl }
5363     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5364     { \tl_set:Nn \l_tmpa_tl { 1 } }
5365     \bool_lazy_or:nnT
5366     { \tl_if_blank_p:V \l_tmpb_tl }
5367     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5368     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5369     \int_compare:nNnT \l_tmpb_tl > #2
5370     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5371     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5372     { \clist_put_right:Nn #1 { ####1 } }
5373   }
5374 }
```

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

```
5375 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5376 {
5377   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5378   {
```

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`).

```
5379     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5380     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5381   }
5382   \ignorespaces
5383 }
```

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

```
5384 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5385 {
5386   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5387   {
5388     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5389     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5390     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5391   }
5392   \ignorespaces
5393 }
```

```

5394 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5395 {

```

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).

```

5396 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5397 {

```

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).

```

5398 \tl_gput_left:Nx \g_@@_pre_code_before_tl
5399 {
5400 \exp_not:N \columncolor [ #1 ]
5401 { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5402 }
5403 }
5404 }

```

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`.

```

5405 \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`.

```

5406 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5407 {
5408 \int_compare:nNnTF \l_@@_first_col_int = 0
5409 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5410 {
5411 \int_compare:nNnTF \c@jCol = 0
5412 {
5413 \int_compare:nNnF \c@iRow = { -1 }
5414 { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5415 }
5416 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5417 }
5418 }

```

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.

```

5419 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5420 {
5421 \int_compare:nNnF \c@iRow = 0
5422 { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
5423 }

```


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.

```

5424 \keys_define:nn { NiceMatrix / Rules }
5425 {
5426   position .int_set:N = \l_@@_position_int ,
5427   position .value_required:n = true ,
5428   start .int_set:N = \l_@@_start_int ,
5429   start .initial:n = 1 ,
5430   end .code:n =
5431     \bool_lazy_or:nnTF
5432       { \tl_if_empty_p:n { #1 } }
5433       { \str_if_eq_p:nn { #1 } { last } }
5434       { \int_set_eq:NN \l_@@_end_int \c@jCol }
5435       { \int_set:Nn \l_@@_end_int { #1 } }
5436 }

```

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.

```

5437 \keys_define:nn { NiceMatrix / RulesBis }
5438 {
5439   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5440   multiplicity .initial:n = 1 ,
5441   dotted .bool_set:N = \l_@@_dotted_bool ,
5442   dotted .initial:n = false ,
5443   dotted .default:n = true ,
5444   color .code:n = \@@_set_CT@arc@:n { #1 } ,
5445   color .value_required:n = true ,
5446   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5447   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`.

```

5448   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5449   tikz .value_required:n = true ,
5450   tikz .initial:n = ,
5451   total-width .dim_set:N = \l_@@_rule_width_dim ,
5452   total-width .value_required:n = true ,
5453   width .meta:n = { total-width = #1 } ,
5454   unknown .code:n = \@@_error:n { Unknow-key-for~RulesBis }
5455 }

```

The vertical rules

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

```

5456 \cs_new_protected:Npn \@@_vline:n #1
5457 {

```

The group is for the options.

```

5458 \group_begin:
5459 \int_zero_new:N \l_@@_end_int
5460 \int_set_eq:NN \l_@@_end_int \c@iRow
5461 \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).

```

5462 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5463 \@@_vline_i:
5464 \group_end:
5465 }

5466 \cs_new_protected:Npn \@@_vline_i:
5467 {
5468 \int_zero_new:N \l_@@_local_start_int
5469 \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`.

```

5470 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5471 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5472 \l_tmpa_tl
5473 {

```

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.

```

5474 \bool_gset_true:N \g_tmpa_bool
5475 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5476 { \@@_test_vline_in_block:nnnnn #1 }
5477 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5478 { \@@_test_vline_in_block:nnnnn #1 }
5479 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5480 { \@@_test_vline_in_stroken_block:nnnn #1 }
5481 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5482 \bool_if:NTF \g_tmpa_bool
5483 {
5484 \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.

```

5485 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5486 }
5487 {
5488 \int_compare:nNnT \l_@@_local_start_int > 0
5489 {
5490 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5491 \@@_vline_ii:
5492 \int_zero:N \l_@@_local_start_int
5493 }
5494 }
5495 }
5496 \int_compare:nNnT \l_@@_local_start_int > 0
5497 {
5498 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5499 \@@_vline_ii:
5500 }
5501 }

```

```

5502 \cs_new_protected:Npn \@@_test_in_corner_v:
5503 {
5504 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } } {

```

```

5505 {
5506   \seq_if_in:NxT
5507   \l_@@_corners_cells_seq
5508   { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5509   { \bool_set_false:N \g_tmpa_bool }
5510 }
5511 {
5512   \seq_if_in:NxT
5513   \l_@@_corners_cells_seq
5514   { \l_tmpa_tl - \l_tmpb_tl }
5515   {
5516     \int_compare:nNnTF \l_tmpb_tl = 1
5517     { \bool_set_false:N \g_tmpa_bool }
5518     {
5519       \seq_if_in:NxT
5520       \l_@@_corners_cells_seq
5521       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5522       { \bool_set_false:N \g_tmpa_bool }
5523     }
5524   }
5525 }
5526 }

5527 \cs_new_protected:Npn \@@_vline_ii:
5528 {
5529   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5530   \bool_if:NTF \l_@@_dotted_bool
5531   \@@_vline_iv:
5532   {
5533     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5534     \@@_vline_iii:
5535     \@@_vline_v:
5536   }
5537 }

```

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

```

5538 \cs_new_protected:Npn \@@_vline_iii:
5539 {
5540   \pgfpicture
5541   \pgfrememberpicturepositiononpagetrue
5542   \pgf@relevantforpicturesizefalse
5543   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5544   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5545   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5546   \dim_set:Nn \l_tmpb_dim
5547   {
5548     \pgf@x
5549     - 0.5 \l_@@_rule_width_dim
5550     +
5551     ( \arrayrulewidth * \l_@@_multiplicity_int
5552       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5553   }
5554   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5555   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5556   \bool_lazy_all:nT
5557   {
5558     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5559     { \cs_if_exist_p:N \CT@drsc@ }
5560     { ! \tl_if_blank_p:V \CT@drsc@ }
5561   }
5562   {
5563     \group_begin:

```

```

5564 \CT@drsc@
5565 \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5566 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5567 \dim_set:Nn \l_@@_tmpd_dim
5568 {
5569     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5570     * ( \l_@@_multiplicity_int - 1 )
5571 }
5572 \pgfpathrectanglecorners
5573 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5574 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5575 \pgfusepath { fill }
5576 \group_end:
5577 }
5578 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5579 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5580 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5581 {
5582     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5583     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5584     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5585     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5586 }
5587 \CT@arc@
5588 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5589 \pgfsetrectcap
5590 \pgfusepathqstroke
5591 \endpgfpicture
5592 }

```

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

```

5593 \cs_new_protected:Npn \@@_vline_iv:
5594 {
5595     \pgfpicture
5596     \pgfrememberpicturepositiononpagetrue
5597     \pgf@relevantforpicturesizefalse
5598     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5599     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5600     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5601     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5602     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5603     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5604     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5605     \CT@arc@
5606     \@@_draw_line:
5607     \endpgfpicture
5608 }

```

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`).

```

5609 \cs_new_protected:Npn \@@_vline_v:
5610 {
5611     \begin {tikzpicture }
5612     \pgfrememberpicturepositiononpagetrue
5613     \pgf@relevantforpicturesizefalse
5614     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5615     \dim_set_eq:NN \l_tmpa_dim \pgf@y
5616     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5617     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5618     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5619     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5620     \exp_args:NV \tikzset \l_@@_tikz_rule_tl

```

```

5621 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5622 ( \l_tmpb_dim , \l_tmpa_dim ) --
5623 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5624 \end { tikzpicture }
5625 }

```

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).

```

5626 \cs_new_protected:Npn \@@_draw_vlines:
5627 {
5628   \int_step_inline:nnn
5629   {
5630     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5631     1 2
5632   }
5633   {
5634     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5635     { \int_eval:n { \c@jCol + 1 } }
5636     \c@jCol
5637   }
5638   {
5639     \tl_if_eq:NnF \l_@@_vlines_clist { all }
5640     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5641     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5642   }
5643 }

```

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}`.

```

5644 \cs_new_protected:Npn \@@_hline:n #1
5645 {

```

The group is for the options.

```

5646   \group_begin:
5647   \int_zero_new:N \l_@@_end_int
5648   \int_set_eq:NN \l_@@_end_int \c@jCol
5649   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5650   \@@_hline_i:
5651   \group_end:
5652 }
5653 \cs_new_protected:Npn \@@_hline_i:
5654 {
5655   \int_zero_new:N \l_@@_local_start_int
5656   \int_zero_new:N \l_@@_local_end_int

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

5657   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5658   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5659   \l_tmpb_tl
5660   {

```

The boolean `\g_tmpa_bool` indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small horizontal rule won't be drawn.

```

5661     \bool_gset_true:N \g_tmpa_bool
5662     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5663     { \@@_test_hline_in_block:nnnnn ##1 }
5664     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq

```

```

5665         { \@@_test_hline_in_block:nnnnn ##1 }
5666     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5667         { \@@_test_hline_in_stroken_block:nnnn ##1 }
5668     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5669     \bool_if:NTF \g_tmpa_bool
5670     {
5671         \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.

```

5672         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5673     }
5674     {
5675         \int_compare:nNnT \l_@@_local_start_int > 0
5676         {
5677             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5678             \@@_hline_ii:
5679             \int_zero:N \l_@@_local_start_int
5680         }
5681     }
5682 }
5683 \int_compare:nNnT \l_@@_local_start_int > 0
5684 {
5685     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5686     \@@_hline_ii:
5687 }
5688 }

```

```

5689 \cs_new_protected:Npn \@@_test_in_corner_h:
5690 {
5691     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5692     {
5693         \seq_if_in:NxT
5694         \l_@@_corners_cells_seq
5695         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5696         { \bool_set_false:N \g_tmpa_bool }
5697     }
5698     {
5699         \seq_if_in:NxT
5700         \l_@@_corners_cells_seq
5701         { \l_tmpa_tl - \l_tmpb_tl }
5702         {
5703             \int_compare:nNnTF \l_tmpa_tl = 1
5704             { \bool_set_false:N \g_tmpa_bool }
5705             {
5706                 \seq_if_in:NxT
5707                 \l_@@_corners_cells_seq
5708                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5709                 { \bool_set_false:N \g_tmpa_bool }
5710             }
5711         }
5712     }
5713 }

```

```

5714 \cs_new_protected:Npn \@@_hline_ii:
5715 {
5716     % \bool_set_false:N \l_@@_dotted_bool
5717     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5718     \bool_if:NTF \l_@@_dotted_bool
5719     \@@_hline_iv:
5720     {
5721         \tl_if_empty:NTF \l_@@_tikz_rule_tl

```

```

5722         \@@_hline_iii:
5723         \@@_hline_v:
5724     }
5725 }

```

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

```

5726 \cs_new_protected:Npn \@@_hline_iii:
5727 {
5728     \pgfpicture
5729     \pgfrememberpicturepositiononpagetrue
5730     \pgf@relevantforpicturesizefalse
5731     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5732     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5733     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5734     \dim_set:Nn \l_tmpb_dim
5735     {
5736         \pgf@y
5737         - 0.5 \l_@@_rule_width_dim
5738         +
5739         ( \arrayrulewidth * \l_@@_multiplicity_int
5740           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5741     }
5742     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5743     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5744     \bool_lazy_all:nT
5745     {
5746         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5747         { \cs_if_exist_p:N \CT@drsc@ }
5748         { ! \tl_if_blank_p:V \CT@drsc@ }
5749     }
5750     {
5751         \group_begin:
5752         \CT@drsc@
5753         \dim_set:Nn \l_@@_tmpd_dim
5754         {
5755             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5756             * ( \l_@@_multiplicity_int - 1 )
5757         }
5758         \pgfpathrectanglecorners
5759         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5760         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5761         \pgfusepathqfill
5762         \group_end:
5763     }
5764     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5765     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5766     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5767     {
5768         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5769         \dim_sub:Nn \l_tmpb_dim \doublerulesep
5770         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5771         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5772     }
5773     \CT@arc@
5774     \pgfsetlinewidth { 1.1 \arrayrulewidth }
5775     \pgfsetrectcap
5776     \pgfusepathqstroke
5777     \endpgfpicture
5778 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

```

5779 \cs_new_protected:Npn \@@_hline_iv:
5780 {
5781   \pgfpicture
5782   \pgfrememberpicturerepositiononpagetrue
5783   \pgf@relevantforpicturesizefalse
5784   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5785   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5786   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
5787   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5788   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5789   \int_compare:nNnT \l_@@_local_start_int = 1
5790   {
5791     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5792     \bool_if:NT \g_@@_NiceArray_bool
5793     { \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.

```

5794   \tl_if_eq:NnF \g_@@_left_delim_tl (
5795     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
5796   )
5797   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5798   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5799   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5800   {
5801     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5802     \bool_if:NT \g_@@_NiceArray_bool
5803     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5804     \tl_if_eq:NnF \g_@@_right_delim_tl )
5805     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5806   }
5807   \CT@arc@
5808   \@@_draw_line:
5809   \endpgfpicture
5810 }

```

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`).

```

5811 \cs_new_protected:Npn \@@_hline_v:
5812 {
5813   \begin { tikzpicture }
5814   \pgfrememberpicturerepositiononpagetrue
5815   \pgf@relevantforpicturesizefalse
5816   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5817   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5818   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5819   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }

```



```

5820 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5821 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5822 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5823 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5824 ( \l_tmpa_dim , \l_tmpb_dim ) --
5825 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5826 \end { tikzpicture }
5827 }

```

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)).

```

5828 \cs_new_protected:Npn \@@_draw_hlines:
5829 {
5830   \int_step_inline:nnn
5831   {
5832     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5833     { 1 2
5834     }
5835     {
5836       \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5837       { \int_eval:n { \c@iRow + 1 } }
5838       \c@iRow
5839     }
5840     {
5841       \tl_if_eq:NnF \l_@@_hlines_clist { all }
5842       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5843       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5844     }
5845   }

```

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

```

5846 \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.

```

5847 \cs_set:Npn \@@_Hline_i:n #1
5848 {
5849   \peek_remove_spaces:n
5850   {
5851     \peek_meaning:NTF \Hline
5852     { \@@_Hline_ii:nn { #1 + 1 } }
5853     { \@@_Hline_iii:n { #1 } }
5854   }
5855 }
5856 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5857 \cs_set:Npn \@@_Hline_iii:n #1
5858 {
5859   \peek_meaning:NTF [
5860   { \@@_Hline_iv:nw { #1 } }
5861   { \@@_Hline_iv:nw { #1 } [ ] }
5862 }
5863 \cs_set:Npn \@@_Hline_iv:nw #1 [ #2 ]
5864 {
5865   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
5866   \skip_vertical:n { \l_@@_rule_width_dim }
5867   \tl_gput_right:Nx \g_@@_pre_code_after_tl
5868   {
5869     \@@_hline:n
5870     {
5871       multiplicity = #1 ,
5872       position = \int_eval:n { \c@iRow + 1 } ,

```

```

5873         total-width = \dim_use:N \l_@@_rule_width_dim ,
5874         #2
5875     }
5876 }
5877 \egroup
5878 }

```

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).

```

5879 \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`).

```

5880 \cs_new_protected:Npn \@@_custom_line:n #1
5881 {
5882     \str_clear_new:N \l_@@_command_str
5883     \str_clear_new:N \l_@@_ccommand_str
5884     \str_clear_new:N \l_@@_letter_str
5885     \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.

```

5886     \bool_lazy_all:nTF
5887     {
5888         { \str_if_empty_p:N \l_@@_letter_str }
5889         { \str_if_empty_p:N \l_@@_command_str }
5890         { \str_if_empty_p:N \l_@@_cccommand_str }
5891     }
5892     { \@@_error:n { No~letter~and~no~command } }
5893     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
5894 }

5895 \keys_define:nn { NiceMatrix / custom-line }
5896 {
5897     letter .str_set:N = \l_@@_letter_str ,
5898     letter .value_required:n = true ,
5899     command .str_set:N = \l_@@_command_str ,
5900     command .value_required:n = true ,
5901     ccommand .str_set:N = \l_@@_cccommand_str ,
5902     ccommand .value_required:n = true ,
5903 }

```

```

5904 \cs_new_protected:Npn \@@_custom_line_i:n #1
5905 {

```

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

```

5906     \bool_set_false:N \l_@@_tikz_rule_bool
5907     \bool_set_false:N \l_@@_dotted_rule_bool
5908     \bool_set_false:N \l_@@_color_bool

```

```

5909 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
5910 \bool_if:NT \l_@@_tikz_rule_bool
5911 {
5912   \IfPackageLoadedTF { tikz }
5913   { }
5914   { \@@_error:n { tikz~in~custom~line~without~tikz } }
5915   \bool_if:NT \l_@@_color_bool
5916   { \@@_error:n { color~in~custom~line~with~tikz } }
5917 }
5918 \bool_if:nT
5919 {
5920   \int_compare_p:nNn \l_@@_multiplicity_int > 1
5921   && \l_@@_dotted_rule_bool
5922 }
5923 { \@@_error:n { key~multiplicity~with~dotted } }
5924 \str_if_empty:NF \l_@@_letter_str
5925 {
5926   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
5927   { \@@_error:n { Several~letters } }
5928   {
5929     \exp_args:NnV \tl_if_in:NnTF
5930     \c_@@_forbidden_letters_str \l_@@_letter_str
5931     { \@@_error:n { Forbidden~letter } }
5932     {

```

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.

```

5933 \keys_define:nx { NiceMatrix / ColumnTypes }
5934 {
5935   \l_@@_letter_str .code:n =
5936   { \@@_v_custom_line:n { \exp_not:n { #1 } } }
5937 }
5938 }
5939 }
5940 }
5941 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
5942 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
5943 }
5944 \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.

```

5945 \keys_define:nn { NiceMatrix / custom-line-bis }
5946 {
5947   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5948   multiplicity .initial:n = 1 ,
5949   multiplicity .value_required:n = true ,
5950   color .code:n = \bool_set_true:N \l_@@_color_bool ,
5951   color .value_required:n = true ,
5952   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5953   tikz .value_required:n = true ,
5954   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5955   dotted .value_forbidden:n = true ,
5956   total-width .code:n = { } ,
5957   total-width .value_required:n = true ,
5958   width .code:n = { } ,
5959   width .value_required:n = true ,
5960   sep-color .code:n = { } ,
5961   sep-color .value_required:n = true ,
5962   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
5963 }

```

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

```

5964 \bool_new:N \l_@@_dotted_rule_bool
5965 \bool_new:N \l_@@_tikz_rule_bool
5966 \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`.

```

5967 \keys_define:nn { NiceMatrix / custom-line-width }
5968 {
5969   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5970   multiplicity .initial:n = 1 ,
5971   multiplicity .value_required:n = true ,
5972   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5973   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5974                       \bool_set_true:N \l_@@_total_width_bool ,
5975   total-width .value_required:n = true ,
5976   width .meta:n = { total-width = #1 } ,
5977   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5978 }

```

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`).

```

5979 \cs_new_protected:Npn \@@_h_custom_line:n #1
5980 {

```

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`.

```

5981   \cs_set:cpn { nicematrix - \l_@@_command_str }
5982   {
5983     \noalign
5984     {
5985       \@@_compute_rule_width:n { #1 }
5986       \skip_vertical:n { \l_@@_rule_width_dim }
5987       \tl_gput_right:Nx \g_@@_pre_code_after_tl
5988       {
5989         \@@_hline:n
5990         {
5991           #1 ,
5992           position = \int_eval:n { \c@iRow + 1 } ,
5993           total-width = \dim_use:N \l_@@_rule_width_dim
5994         }
5995       }
5996     }
5997   }
5998   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
5999 }
6000 \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`).

```

6001 \cs_new_protected:Npn \@@_c_custom_line:n #1
6002 {

```

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

```

6003   \exp_args:Nc \NewExpandableDocumentCommand
6004   { nicematrix - \l_@@_ccommand_str }
6005   { 0 { } m }

```

```

6006 {
6007   \noalign
6008   {
6009     \@@_compute_rule_width:n { #1 , ##1 }
6010     \skip_vertical:n { \l_@@_rule_width_dim }
6011     \clist_map_inline:nn
6012       { ##2 }
6013     { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6014   }
6015 }
6016 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6017 }

```

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*.

```

6018 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6019 {
6020   \str_if_in:nnTF { #2 } { - }
6021   { \@@_cut_on_hyphen:w #2 \q_stop }
6022   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6023   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6024   {
6025     \@@_hline:n
6026     {
6027       #1 ,
6028       start = \l_tmpa_tl ,
6029       end = \l_tmpb_tl ,
6030       position = \int_eval:n { \c@iRow + 1 } ,
6031       total-width = \dim_use:N \l_@@_rule_width_dim
6032     }
6033   }
6034 }
6035 \cs_generate_variant:Nn \@@_c_custom_line:nn { n V }
6036 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6037 {
6038   \bool_set_false:N \l_@@_tikz_rule_bool
6039   \bool_set_false:N \l_@@_total_width_bool
6040   \bool_set_false:N \l_@@_dotted_rule_bool
6041   \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6042   \bool_if:NF \l_@@_total_width_bool
6043   {
6044     \bool_if:NTF \l_@@_dotted_rule_bool
6045     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6046     {
6047       \bool_if:NF \l_@@_tikz_rule_bool
6048       {
6049         \dim_set:Nn \l_@@_rule_width_dim
6050         {
6051           \arrayrulewidth * \l_@@_multiplicity_int
6052           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6053         }
6054       }
6055     }
6056   }
6057 }
6058 \cs_new_protected:Npn \@@_v_custom_line:n #1
6059 {
6060   \@@_compute_rule_width:n { #1 }

```

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

```

6061   \tl_gput_right:Nx \g_@@_preamble_tl
6062   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6063   \tl_gput_right:Nx \g_@@_pre_code_after_tl

```

```

6064     {
6065         \@@_vline:n
6066         {
6067             #1 ,
6068             position = \int_eval:n { \c@jCol + 1 } ,
6069             total-width = \dim_use:N \l_@@_rule_width_dim
6070         }
6071     }
6072 }
6073 \@@_custom_line:n
6074 { 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.

```

6075 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6076 {
6077     \bool_lazy_all:nT
6078     {
6079         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6080         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6081         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6082         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6083     }
6084     { \bool_gset_false:N \g_tmpa_bool }
6085 }

```

The same for vertical rules.

```

6086 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6087 {
6088     \bool_lazy_all:nT
6089     {
6090         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6091         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6092         { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6093         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6094     }
6095     { \bool_gset_false:N \g_tmpa_bool }
6096 }
6097 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6098 {
6099     \bool_lazy_all:nT
6100     {
6101         {
6102             ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6103             || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6104         }
6105         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6106         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6107     }
6108     { \bool_gset_false:N \g_tmpa_bool }
6109 }
6110 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6111 {
6112     \bool_lazy_all:nT
6113     {
6114         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6115         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6116         {

```

```

6117         ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6118         || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6119     }
6120 }
6121 { \bool_gset_false:N \g_tmpa_bool }
6122 }

```

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.

```

6123 \cs_new_protected:Npn \@@_compute_corners:
6124 {

```

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.

```

6125     \seq_clear_new:N \l_@@_corners_cells_seq
6126     \clist_map_inline:Nn \l_@@_corners_clist
6127     {
6128         \str_case:nnF { ##1 }
6129         {
6130             { NW }
6131             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6132             { NE }
6133             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6134             { SW }
6135             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6136             { SE }
6137             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6138         }
6139         { \@@_error:nn { bad~corner } { ##1 } }
6140     }

```

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

```

6141     \seq_if_empty:NF \l_@@_corners_cells_seq
6142     {

```

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.

```

6143         \tl_gput_right:Nx \g_@@_aux_tl
6144         {
6145             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6146             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6147         }
6148     }
6149 }

```

“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.

```

6150 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6151 {

```

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.

```

6152 \bool_set_false:N \l_tmpa_bool
6153 \int_zero_new:N \l_@@_last_empty_row_int
6154 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6155 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6156 {
6157   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6158   \bool_lazy_or:nnTF
6159   {
6160     \cs_if_exist_p:c
6161     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6162   }
6163   \l_tmpb_bool
6164   { \bool_set_true:N \l_tmpa_bool }
6165   {
6166     \bool_if:NF \l_tmpa_bool
6167     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6168   }
6169 }

```

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

```

6170 \bool_set_false:N \l_tmpa_bool
6171 \int_zero_new:N \l_@@_last_empty_column_int
6172 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6173 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6174 {
6175   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6176   \bool_lazy_or:nnTF
6177   \l_tmpb_bool
6178   {
6179     \cs_if_exist_p:c
6180     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6181   }
6182   { \bool_set_true:N \l_tmpa_bool }
6183   {
6184     \bool_if:NF \l_tmpa_bool
6185     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6186   }
6187 }

```

Now, we loop over the rows.

```

6188 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6189 {

```

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

```

6190 \bool_set_false:N \l_tmpa_bool
6191 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6192 {
6193   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6194   \bool_lazy_or:nnTF
6195   \l_tmpb_bool
6196   {
6197     \cs_if_exist_p:c
6198     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6199   }
6200   { \bool_set_true:N \l_tmpa_bool }
6201   {
6202     \bool_if:NF \l_tmpa_bool

```



```

6203         {
6204             \int_set:Nn \l_@@_last_empty_column_int { ####1 }
6205             \seq_put_right:Nn
6206                 \l_@@_corners_cells_seq
6207                 { ##1 - ####1 }
6208         }
6209     }
6210 }
6211 }
6212 }

```

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`).

```

6213 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6214 {
6215     \int_set:Nn \l_tmpa_int { #1 }
6216     \int_set:Nn \l_tmpb_int { #2 }
6217     \bool_set_false:N \l_tmpb_bool
6218     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6219         { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6220 }
6221 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6222 {
6223     \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6224     {
6225         \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6226         {
6227             \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6228             {
6229                 \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6230                 { \bool_set_true:N \l_tmpb_bool }
6231             }
6232         }
6233     }
6234 }

```

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.

```

6235 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6236 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6237 {
6238     auto-columns-width .code:n =
6239     {
6240         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6241         \dim_gzero_new:N \g_@@_max_cell_width_dim
6242         \bool_set_true:N \l_@@_auto_columns_width_bool
6243     }
6244 }

```

```

6245 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6246 {
6247   \int_gincr:N \g_@@_NiceMatrixBlock_int
6248   \dim_zero:N \l_@@_columns_width_dim
6249   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6250   \bool_if:NT \l_@@_block_auto_columns_width_bool
6251   {
6252     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6253     {
6254       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
6255       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6256     }
6257   }
6258 }

```

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).

```

6259 {
6260   \bool_if:NT \l_@@_block_auto_columns_width_bool
6261   {
6262     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6263     \iow_shipout:Nx \@mainaux
6264     {
6265       \cs_gset:cpn
6266       { @@_max_ cell_ width_ \int_use:N \g_@@_NiceMatrixBlock_int }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

6267       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6268     }
6269     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6270   }
6271 }

```

25 The extra nodes

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

```

6272 \cs_generate_variant:Nn \dim_min:nn { v n }
6273 \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).

```

6274 \cs_new_protected:Npn \@@_create_extra_nodes:
6275 {
6276   \bool_if:nTF \l_@@_medium_nodes_bool
6277   {
6278     \bool_if:nTF \l_@@_large_nodes_bool
6279     \@@_create_medium_and_large_nodes:
6280     \@@_create_medium_nodes:
6281   }
6282   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6283 }

```

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`.

```

6284 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6285 {
6286   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6287   {
6288     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6289     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6290     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6291     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6292   }
6293   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6294   {
6295     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6296     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6297     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6298     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6299   }

```

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

```

6300   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6301   {
6302     \int_step_variable:nnNn
6303     \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.

```

6304     {
6305       \cs_if_exist:cT
6306       { 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`.

```

6307     {
6308       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6309       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6310       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6311       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6312       {
6313         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6314         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6315       }

```

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`.

```

6316       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6317       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6318       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6319       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6320       {
6321         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }

```

```

6322         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6323     }
6324 }
6325 }
6326 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6327 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6328 {
6329     \dim_compare:nNnT
6330     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6331     {
6332         \@@_qpoint:n { row - \@@_i: - base }
6333         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6334         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6335     }
6336 }
6337 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6338 {
6339     \dim_compare:nNnT
6340     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6341     {
6342         \@@_qpoint:n { col - \@@_j: }
6343         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@x
6344         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@x
6345     }
6346 }
6347 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6348 \cs_new_protected:Npn \@@_create_medium_nodes:
6349 {
6350     \pgfpicture
6351     \pgfrememberpicturepositiononpagetrue
6352     \pgf@relevantforpicturesizefalse
6353     \@@_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”.

```

6354     \tl_set:Nn \l_@@_suffix_tl { -medium }
6355     \@@_create_nodes:
6356     \endpgfpicture
6357 }

```

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:`.

```

6358 \cs_new_protected:Npn \@@_create_large_nodes:
6359 {
6360     \pgfpicture
6361     \pgfrememberpicturepositiononpagetrue
6362     \pgf@relevantforpicturesizefalse
6363     \@@_computations_for_medium_nodes:
6364     \@@_computations_for_large_nodes:
6365     \tl_set:Nn \l_@@_suffix_tl { - large }
6366     \@@_create_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6367 \endpgfpicture
6368 }
6369 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6370 {
6371 \pgfpicture
6372 \pgfrememberpicturepositiononpagetrue
6373 \pgf@relevantforpicturesizefalse
6374 \@@_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”.

```

6375 \tl_set:Nn \l_@@_suffix_tl { - medium }
6376 \@@_create_nodes:
6377 \@@_computations_for_large_nodes:
6378 \tl_set:Nn \l_@@_suffix_tl { - large }
6379 \@@_create_nodes:
6380 \endpgfpicture
6381 }

```

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

6382 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6383 {
6384 \int_set:Nn \l_@@_first_row_int 1
6385 \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`.

```

6386 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6387 {
6388 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6389 {
6390 (
6391 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6392 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6393 )
6394 / 2
6395 }
6396 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6397 { l_@@_row _ \@@_i: _ min_dim }
6398 }
6399 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6400 {
6401 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6402 {
6403 (
6404 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6405 \dim_use:c
6406 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6407 )
6408 / 2
6409 }
6410 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6411 { l_@@_column _ \@@_j: _ max _ dim }
6412 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6413 \dim_sub:cn
6414 { l_@@_column _ 1 _ min _ dim }
6415 \l_@@_left_margin_dim
6416 \dim_add:cn
6417 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6418 \l_@@_right_margin_dim
6419 }

```

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`).

```

6420 \cs_new_protected:Npn \@@_create_nodes:
6421 {
6422   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6423   {
6424     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6425     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6426       \@@_pgf_rect_node:nnnnn
6427       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6428       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6429       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6430       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6431       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6432       \str_if_empty:NF \l_@@_name_str
6433       {
6434         \pgfnodealias
6435         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6436         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6437       }
6438     }
6439   }

```

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 .

```

6440   \cs_if_exist_use:NF
6441   \seq_map_pairwise_function:NNN
6442   \seq_mapthread_function:NNN
6443   \g_@@_multicolumn_cells_seq
6444   \g_@@_multicolumn_sizes_seq
6445   \@@_node_for_multicolumn:nn
6446 }

```

```

6447 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6448 {
6449   \cs_set_nopar:Npn \@@_i: { #1 }
6450   \cs_set_nopar:Npn \@@_j: { #2 }
6451 }

```

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”).

```

6452 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6453 {
6454   \@@_extract_coords_values: #1 \q_stop
6455   \@@_pgf_rect_node:nnnnn
6456   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6457   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6458   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6459   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
6460   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6461   \str_if_empty:NF \l_@@_name_str
6462   {
6463     \pgfnodealias
6464     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6465     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}

```

```

6466     }
6467 }

```

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.

```

6468 \keys_define:nn { NiceMatrix / Block / FirstPass }
6469 {
6470   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6471   l .value_forbidden:n = true ,
6472   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6473   r .value_forbidden:n = true ,
6474   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6475   c .value_forbidden:n = true ,
6476   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6477   L .value_forbidden:n = true ,
6478   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6479   R .value_forbidden:n = true ,
6480   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6481   C .value_forbidden:n = true ,
6482   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6483   t .value_forbidden:n = true ,
6484   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6485   b .value_forbidden:n = true ,
6486   color .tl_set:N = \l_@@_color_tl ,
6487   color .value_required:n = true ,
6488   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6489   respect-arraystretch .default:n = true ,
6490 }

```

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.

```

6491 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
6492 {

```

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).

```

6493   \peek_remove_spaces:n
6494   {
6495     \tl_if_blank:nTF { #2 }
6496     { \@@_Block_i 1-1 \q_stop }
6497     {
6498       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6499       \@@_Block_i_czech \@@_Block_i
6500       #2 \q_stop
6501     }
6502     { #1 } { #3 } { #4 }
6503   }
6504 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6505 \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`.

```

6506 {
6507   \char_set_catcode_active:N -
6508   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6509 }

```

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.

```

6510 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6511 {

```

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).

```

6512   \bool_lazy_or:nnTF
6513     { \tl_if_blank_p:n { #1 } }
6514     { \str_if_eq_p:nn { #1 } { * } }
6515     { \int_set:Nn \l_tmpa_int { 100 } }
6516     { \int_set:Nn \l_tmpa_int { #1 } }
6517   \bool_lazy_or:nnTF
6518     { \tl_if_blank_p:n { #2 } }
6519     { \str_if_eq_p:nn { #2 } { * } }
6520     { \int_set:Nn \l_tmpb_int { 100 } }
6521     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6522   \int_compare:nNnTF \l_tmpb_int = 1
6523   {
6524     \str_if_empty:NTF \l_@@_hpos_cell_str
6525     { \str_set:Nn \l_@@_hpos_block_str c }
6526     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6527   }
6528   { \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.

```

6529   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6530   \tl_set:Nx \l_tmpa_tl
6531   {
6532     { \int_use:N \c@iRow }
6533     { \int_use:N \c@jCol }
6534     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6535     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6536   }

```

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).

```

6537   \bool_if:nTF
6538   {
6539     (
6540       \int_compare_p:nNn { \l_tmpa_int } = 1
6541       ||

```



```

6542     \int_compare:nNn { \l_tmpb_int } = 1
6543   )
6544   && ! \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.

```

6545     && ! \l_@@_X_column_bool
6546   }
6547   { \exp_args:Nxx \@@_Block_iv:nnnnn }
6548   { \exp_args:Nxx \@@_Block_v:nnnnn }
6549   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6550 }

```

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).

```

6551 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6552 {
6553   \int_gincr:N \g_@@_block_box_int
6554   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6555   {
6556     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6557     {
6558       \@@_actually_diagbox:nnnnnn
6559       { \int_use:N \c@iRow }
6560       { \int_use:N \c@jCol }
6561       { \int_eval:n { \c@iRow + #1 - 1 } }
6562       { \int_eval:n { \c@jCol + #2 - 1 } }
6563       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6564     }
6565   }
6566   \box_gclear_new:c
6567   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
6568   \hbox_gset:cn
6569   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
6570   {

```

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}).

```

6571     \tl_if_empty:NTF \l_@@_color_tl
6572     { \int_compare:nNnT { #2 } = 1 \set@color }
6573     { \@@_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.

```

6574     \int_compare:nNnT { #1 } = 1
6575     {
6576       \int_compare:nNnTF \c@iRow = 0
6577       \l_@@_code_for_first_row_tl
6578       {
6579         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6580         \l_@@_code_for_last_row_tl
6581       }
6582       \g_@@_row_style_tl
6583     }

```

```

6584 \group_begin:
6585 \bool_if:NF \l_@@_respect_arraystretch_bool
6586 { \cs_set:Npn \arraystretch { 1 } }
6587 \dim_zero:N \extrarowheight
6588 #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).

```

6589 \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
6590 \bool_if:NTF \l_@@_NiceTabular_bool
6591 {
6592   \bool_lazy_all:nTF
6593   {
6594     { \int_compare_p:nNn { #2 } = 1 }
6595     { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6596     { ! \g_@@_rotate_bool }
6597   }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

6598   {
6599     \use:x
6600     {
6601       \exp_not:N \begin { minipage }%
6602       [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6603       { \l_@@_col_width_dim }
6604       \str_case:Vn \l_@@_hpos_block_str
6605       {
6606         c \centering
6607         r \raggedleft
6608         l \raggedright
6609       }
6610     }
6611     #5
6612     \end { minipage }
6613   }
6614   {
6615     \use:x
6616     {
6617       \exp_not:N \begin { tabular }%
6618       [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6619       { @ { } \l_@@_hpos_block_str @ { } }
6620     }
6621     #5
6622     \end { tabular }
6623   }
6624 }
6625 {
6626   \c_math_toggle_token
6627   \use:x
6628   {
6629     \exp_not:N \begin { array }%
6630     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6631     { @ { } \l_@@_hpos_block_str @ { } }
6632   }
6633   #5
6634   \end { array }
6635   \c_math_toggle_token
6636 }
6637 \group_end:
6638 }
6639 \bool_if:NT \g_@@_rotate_bool

```

```

6640 {
6641   \box_grotate:cn
6642   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6643   { 90 }
6644   \bool_gset_false:N \g_@@_rotate_bool
6645 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

6646 \int_compare:nNnT { #2 } = 1
6647 {
6648   \dim_gset:Nn \g_@@_blocks_wd_dim
6649   {
6650     \dim_max:nn
6651     \g_@@_blocks_wd_dim
6652     {
6653       \box_wd:c
6654       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6655     }
6656   }
6657 }

```

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.

```

6658 \int_compare:nNnT { #1 } = 1
6659 {
6660   \dim_gset:Nn \g_@@_blocks_ht_dim
6661   {
6662     \dim_max:nn
6663     \g_@@_blocks_ht_dim
6664     {
6665       \box_ht:c
6666       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6667     }
6668   }
6669   \dim_gset:Nn \g_@@_blocks_dp_dim
6670   {
6671     \dim_max:nn
6672     \g_@@_blocks_dp_dim
6673     {
6674       \box_dp:c
6675       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6676     }
6677   }
6678 }
6679 \seq_gput_right:Nx \g_@@_blocks_seq
6680 {
6681   \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.

```

6682 { \exp_not:n { #3 } , \l_@@_hpos_block_str }
6683 {
6684   \box_use_drop:c
6685   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6686 }
6687 }
6688 }

```

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.

```

6689 \cs_new_protected:Npn \l_@@_Block_v:nnnnn #1 #2 #3 #4 #5
6690 {
6691   \seq_gput_right:Nx \g_@@_blocks_seq
6692   {
6693     \l_tmpa_tl
6694     { \exp_not:n { #3 } }
6695     {
6696       \bool_if:NTF \l_@@_NiceTabular_bool
6697       {
6698         \group_begin:
6699         \bool_if:NF \l_@@_respect_arraystretch_bool
6700         { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6701         \exp_not:n
6702         {
6703           \dim_zero:N \extrarowheight
6704           #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).

```

6705         \bool_if:NT \g_@@_rotate_bool
6706         { \str_set:Nn \l_@@_hpos_block_str c }
6707         \use:x
6708         {
6709           \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
6710           { @ { } \l_@@_hpos_block_str @ { } }
6711         }
6712         #5
6713         \end { tabular }
6714       }
6715     \group_end:
6716   }
6717   {
6718     \group_begin:
6719     \bool_if:NF \l_@@_respect_arraystretch_bool
6720     { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6721     \exp_not:n
6722     {
6723       \dim_zero:N \extrarowheight
6724       #4
6725       \bool_if:NT \g_@@_rotate_bool
6726       { \str_set:Nn \l_@@_hpos_block_str c }
6727       \c_math_toggle_token
6728       \use:x
6729       {
6730         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
6731         { @ { } \l_@@_hpos_block_str @ { } }
6732       }
6733       #5
6734       \end { array }
6735       \c_math_toggle_token
6736     }
6737     \group_end:
6738   }
6739 }
6740 }
6741 }

```

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).

```

6742 \keys_define:nn { NiceMatrix / Block / SecondPass }
6743 {
6744   tikz .code:n =
6745     \IfPackageLoadedTF { tikz }
6746       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6747       { \@@_error:n { tikz~key~without~tikz } } ,
6748   tikz .value_required:n = true ,
6749   fill .code:n =
6750     \tl_set_rescan:Nnn
6751       \l_@@_fill_tl
6752       { \char_set_catcode_other:N ! }
6753       { #1 } ,
6754   fill .value_required:n = true ,
6755   draw .code:n =
6756     \tl_set_rescan:Nnn
6757       \l_@@_draw_tl
6758       { \char_set_catcode_other:N ! }
6759       { #1 } ,
6760   draw .default:n = default ,
6761   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6762   rounded-corners .default:n = 4 pt ,
6763   color .code:n =
6764     \@@_color:n { #1 }
6765     \tl_set_rescan:Nnn
6766       \l_@@_draw_tl
6767       { \char_set_catcode_other:N ! }
6768       { #1 } ,
6769   color .value_required:n = true ,
6770   borders .clist_set:N = \l_@@_borders_clist ,
6771   borders .value_required:n = true ,
6772   hvlines .meta:n = { vlines , hlines } ,
6773   vlines .bool_set:N = \l_@@_vlines_block_bool ,
6774   vlines .default:n = true ,
6775   hlines .bool_set:N = \l_@@_hlines_block_bool ,
6776   hlines .default:n = true ,
6777   line-width .dim_set:N = \l_@@_line_width_dim ,
6778   line-width .value_required:n = true ,
6779   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6780   l .value_forbidden:n = true ,
6781   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6782   r .value_forbidden:n = true ,
6783   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6784   c .value_forbidden:n = true ,
6785   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6786     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6787   L .value_forbidden:n = true ,
6788   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6789     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6790   R .value_forbidden:n = true ,
6791   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6792     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6793   C .value_forbidden:n = true ,
6794   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6795   t .value_forbidden:n = true ,
6796   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6797   T .value_forbidden:n = true ,
6798   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6799   b .value_forbidden:n = true ,
6800   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6801   B .value_forbidden:n = true ,
6802   v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
6803   v-center .value_forbidden:n = true ,
6804   name .tl_set:N = \l_@@_block_name_str ,

```

```

6805     name .value_required:n = true ,
6806     name .initial:n = ,
6807     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6808     respect-arraystretch .default:n = true ,
6809     transparent .bool_set:N = \l_@@_transparent_bool ,
6810     transparent .default:n = true ,
6811     transparent .initial:n = false ,
6812     unknown .code:n = \@@_error:n { Unknown-key-for-Block }
6813 }

```

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.

```

6814 \cs_new_protected:Npn \@@_draw_blocks:
6815 {
6816     \cs_set_eq:NN \ialign \@@_old_ialign:
6817     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
6818 }
6819 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6820 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6821     \int_zero_new:N \l_@@_last_row_int
6822     \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.

```

6823     \int_compare:nNnTF { #3 } > { 99 }
6824     { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
6825       { \int_set:Nn \l_@@_last_row_int { #3 } }
6826     \int_compare:nNnTF { #4 } > { 99 }
6827     { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
6828       { \int_set:Nn \l_@@_last_col_int { #4 } }
6829     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6830     {
6831         \int_compare:nTF
6832         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6833         {
6834             \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
6835             \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
6836             \@@_msg_redirect_name:nn { columns-not-used } { none }
6837         }
6838         { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6839     }
6840     {
6841         \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6842         { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6843         { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6844     }
6845 }

```

`#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

```

6846 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6847 {

```

The group is for the keys.

```

6848 \group_begin:
6849 \int_compare:nNnT { #1 } = { #3 }
6850 { \str_set:Nn \l_@@_vpos_of_block_str { t } }
6851 \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
6852 \bool_if:NT \l_@@_vlines_block_bool
6853 {
6854   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6855   {
6856     \@@_vlines_block:nnn
6857     { \exp_not:n { #5 } }
6858     { #1 - #2 }
6859     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6860   }
6861 }
6862 \bool_if:NT \l_@@_hlines_block_bool
6863 {
6864   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6865   {
6866     \@@_hlines_block:nnn
6867     { \exp_not:n { #5 } }
6868     { #1 - #2 }
6869     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6870   }
6871 }
6872 \bool_if:nF
6873 {
6874   \l_@@_transparent_bool
6875   || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
6876 }
6877 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

6878 \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6879 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6880 }

6881 \bool_lazy_and:nnT
6882 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
6883 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
6884 { \@@_error:n { hlines-with-color } }

6885 \tl_if_empty:NF \l_@@_draw_tl
6886 {
6887   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6888   {
6889     \@@_stroke_block:nnn
6890     { \exp_not:n { #5 } }
6891     { #1 - #2 }
6892     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6893   }
6894   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6895   { { #1 } { #2 } { #3 } { #4 } }
6896 }

6897 \clist_if_empty:NF \l_@@_borders_clist
6898 {
6899   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6900   {
6901     \@@_stroke_borders_block:nnn
6902     { \exp_not:n { #5 } }

```

```

6903         { #1 - #2 }
6904         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6905     }
6906 }
6907 \tl_if_empty:NF \l_@@_fill_tl
6908 {
6909     \tl_gput_right:Nx \g_@@_pre_code_before_tl
6910     {
6911         \exp_not:N \roundedrectanglecolor
6912         \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
6913             { \l_@@_fill_tl }
6914             { { \l_@@_fill_tl } }
6915             { #1 - #2 }
6916             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6917             { \dim_use:N \l_@@_rounded_corners_dim }
6918         }
6919     }
6920 \seq_if_empty:NF \l_@@_tikz_seq
6921 {
6922     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6923     {
6924         \@@_block_tikz:nnnnn
6925         { #1 }
6926         { #2 }
6927         { \int_use:N \l_@@_last_row_int }
6928         { \int_use:N \l_@@_last_col_int }
6929         { \seq_use:Nn \l_@@_tikz_seq { , } }
6930     }
6931 }
6932 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6933 {
6934     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6935     {
6936         \@@_actually_diagbox:nnnnnn
6937         { #1 }
6938         { #2 }
6939         { \int_use:N \l_@@_last_row_int }
6940         { \int_use:N \l_@@_last_col_int }
6941         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6942     }
6943 }
6944 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6945 \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.

```

6946 \pgfpicture
6947 \pgfrememberpicturepositiononpagetrue
6948 \pgf@relevantforpicturesizefalse
6949 \@@_qpoint:n { row - #1 }
6950 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6951 \@@_qpoint:n { col - #2 }
6952 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6953 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6954 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6955 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6956 \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.

```

6957 \@@_pgf_rect_node:nnnnn
6958 { \@@_env: - #1 - #2 - block }
6959 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6960 \str_if_empty:NF \l_@@_block_name_str
6961 {
6962   \pgfnodealias
6963   { \@@_env: - \l_@@_block_name_str }
6964   { \@@_env: - #1 - #2 - block }
6965   \str_if_empty:NF \l_@@_name_str
6966   {
6967     \pgfnodealias
6968     { \l_@@_name_str - \l_@@_block_name_str }
6969     { \@@_env: - #1 - #2 - block }
6970   }
6971 }

```

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.

```

6972 \bool_if:NF \l_@@_hpos_of_block_cap_bool
6973 {
6974   \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.

```

6975 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6976 {

```

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.

```

6977 \cs_if_exist:cT
6978 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6979 {
6980   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6981   {
6982     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6983     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6984   }
6985 }
6986 }

```

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.

```

6987     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6988     {
6989         \@@_qpoint:n { col - #2 }
6990         \dim_set_eq:NN \l_tmpb_dim \pgf@x
6991     }
6992     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6993     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6994     {
6995         \cs_if_exist:cT
6996         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6997         {
6998             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6999             {
7000                 \pgfpointanchor
7001                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7002                 { east }
7003                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7004             }
7005         }
7006     }
7007     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7008     {
7009         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7010         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7011     }
7012     \@@_pgf_rect_node:nnnnn
7013     { \@@_env: - #1 - #2 - block - short }
7014     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7015 }

```

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.

```

7016     \bool_if:NT \l_@@_medium_nodes_bool
7017     {
7018         \@@_pgf_rect_node:nnn
7019         { \@@_env: - #1 - #2 - block - medium }
7020         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7021         {
7022             \pgfpointanchor
7023             { \@@_env:
7024                 - \int_use:N \l_@@_last_row_int
7025                 - \int_use:N \l_@@_last_col_int - medium
7026             }
7027             { south~east }
7028         }
7029     }

```

Now, we will put the label of the block.

```

7030     \bool_lazy_any:nTF
7031     {
7032         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7033         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7034         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7035     }
7036     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7037     \int_compare:nNnT { #2 } = 0
7038     { \str_set:Nn \l_@@_hpos_block_str r }
7039     \bool_if:nT \g_@@_last_col_found_bool

```

```

7040     {
7041         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7042         { \str_set:Nn \l_@@_hpos_block_str 1 }
7043     }
7044 \tl_set:Nx \l_tmpa_tl
7045 {
7046     \str_case:Vn \l_@@_vpos_of_block_str
7047     {
7048         c {
7049             \str_case:Vn \l_@@_hpos_block_str
7050             {
7051                 c { center }
7052                 l { west }
7053                 r { east }
7054             }
7055         }
7056     }
7057     T {
7058         \str_case:Vn \l_@@_hpos_block_str
7059         {
7060             c { north }
7061             l { north-west }
7062             r { north-east }
7063         }
7064     }
7065 }
7066 B {
7067     \str_case:Vn \l_@@_hpos_block_str
7068     {
7069         c { south }
7070         l { south-west }
7071         r { south-east }
7072     }
7073 }
7074 }
7075 }
7076 }
7077 \pgftransformshift
7078 {
7079     \pgfpointanchor
7080     {
7081         \@@_env: - #1 - #2 - block
7082         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7083     }
7084     { \l_tmpa_tl }
7085 }
7086 \pgfset
7087 {
7088     inner~xsep = \c_zero_dim ,
7089     inner~ysep = \l_@@_block_ysep_dim
7090 }
7091 \pgfnode
7092 { rectangle }
7093 { \l_tmpa_tl }
7094 { \box_use_drop:N \l_@@_cell_box } { } { }
7095 }
7096 {
7097     \pgfextracty \l_tmpa_dim
7098     {
7099         \@@_qpoint:n
7100         {
7101             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }

```

```

7102         - base
7103     }
7104 }
7105 \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.

```

7106 \pgfpointanchor
7107 {
7108     \@@_env: - #1 - #2 - block
7109     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7110 }
7111 {
7112     \str_case:Vn \l_@@_hpos_block_str
7113     {
7114         c { center }
7115         l { west }
7116         r { east }
7117     }
7118 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7119 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7120 \pgfset { inner~sep = \c_zero_dim }
7121 \pgfnode
7122 { rectangle }
7123 {
7124     \str_case:Vn \l_@@_hpos_block_str
7125     {
7126         c { base }
7127         l { base~west }
7128         r { base~east }
7129     }
7130 }
7131 { \box_use_drop:N \l_@@_cell_box } { } { }
7132 }
7133 \endpgfpicture
7134 \group_end:
7135 }

```

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).

```

7136 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7137 {
7138     \group_begin:
7139     \tl_clear:N \l_@@_draw_tl
7140     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7141     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7142     \pgfpicture
7143     \pgfrememberpicturepositiononpagetrue
7144     \pgf@relevantforpicturesizefalse
7145     \tl_if_empty:NF \l_@@_draw_tl
7146     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7147     \str_if_eq:VnTF \l_@@_draw_tl { default }
7148     { \CT@arc@ }
7149     { \@@_color:V \l_@@_draw_tl }
7150 }
7151 \pgfsetcornersarced
7152 {
7153     \pgfpoint

```

```

7154         { \dim_use:N \l_@@_rounded_corners_dim }
7155         { \dim_use:N \l_@@_rounded_corners_dim }
7156     }
7157 \@@_cut_on_hyphen:w #2 \q_stop
7158 \bool_lazy_and:nnT
7159 { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7160 { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7161 {
7162     \@@_qpoint:n { row - \l_tmpa_tl }
7163     \dim_set:Nn \l_tmpb_dim { \pgf@y }
7164     \@@_qpoint:n { col - \l_tmpb_tl }
7165     \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
7166     \@@_cut_on_hyphen:w #3 \q_stop
7167     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7168     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7169     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7170     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7171     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7172     \dim_set:Nn \l_tmpa_dim { \pgf@y }
7173     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7174     \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
7175     \pgfpathrectanglecorners
7176     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7177     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7178     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7179     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7180     { \pgfusepathqstroke }
7181     { \pgfusepath { stroke } }
7182 }
7183 \endpgfpicture
7184 \group_end:
7185 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7186 \keys_define:nn { NiceMatrix / BlockStroke }
7187 {
7188     color .tl_set:N = \l_@@_draw_tl ,
7189     draw .tl_set:N = \l_@@_draw_tl ,
7190     draw .default:n = default ,
7191     line-width .dim_set:N = \l_@@_line_width_dim ,
7192     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7193     rounded-corners .default:n = 4 pt
7194 }

```

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).

```

7195 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7196 {
7197     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7198     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7199     \@@_cut_on_hyphen:w #2 \q_stop
7200     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7201     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7202     \@@_cut_on_hyphen:w #3 \q_stop
7203     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7204     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7205     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7206     {
7207         \use:x
7208         {
7209             \@@_vline:n
7210             {

```

```

7211         position = ##1 ,
7212         start = \l_@@_tmpc_tl ,
7213         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7214         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7215     }
7216 }
7217 }
7218 }
7219 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7220 {
7221     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7222     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7223     \@@_cut_on_hyphen:w #2 \q_stop
7224     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7225     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7226     \@@_cut_on_hyphen:w #3 \q_stop
7227     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7228     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7229     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7230     {
7231         \use:x
7232         {
7233             \@@_hline:n
7234             {
7235                 position = ##1 ,
7236                 start = \l_@@_tmpd_tl ,
7237                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
7238                 total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7239             }
7240         }
7241     }
7242 }

```

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).

```

7243 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7244 {
7245     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7246     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7247     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7248     { \@@_error:n { borders~forbidden } }
7249     {
7250         \tl_clear_new:N \l_@@_borders_tikz_tl
7251         \keys_set:nV
7252         { NiceMatrix / OnlyForTikzInBorders }
7253         \l_@@_borders_clist
7254         \@@_cut_on_hyphen:w #2 \q_stop
7255         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7256         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7257         \@@_cut_on_hyphen:w #3 \q_stop
7258         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7259         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7260         \@@_stroke_borders_block_i:
7261     }
7262 }
7263 \hook_gput_code:nnn { begindocument } { . }
7264 {
7265     \cs_new_protected:Npx \@@_stroke_borders_block_i:
7266     {
7267         \c_@@_pgfortikzpicture_tl
7268         \@@_stroke_borders_block_ii:

```

```

7269         \c_@@_endpgfortikzpicture_tl
7270     }
7271 }

7272 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7273 {
7274     \pgfrememberpicturepositiononpagetrue
7275     \pgf@relevantforpicturesizefalse
7276     \CT@arc@
7277     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7278     \clist_if_in:NnT \l_@@_borders_clist { right }
7279     { \@@_stroke_vertical:n \l_tmpb_tl }
7280     \clist_if_in:NnT \l_@@_borders_clist { left }
7281     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7282     \clist_if_in:NnT \l_@@_borders_clist { bottom }
7283     { \@@_stroke_horizontal:n \l_tmpa_tl }
7284     \clist_if_in:NnT \l_@@_borders_clist { top }
7285     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7286 }

7287 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7288 {
7289     tikz .code:n =
7290         \cs_if_exist:NTF \tikzpicture
7291         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7292         { \@@_error:n { tikz-in~borders~without~tikz } } ,
7293     tikz .value_required:n = true ,
7294     top .code:n = ,
7295     bottom .code:n = ,
7296     left .code:n = ,
7297     right .code:n = ,
7298     unknown .code:n = \@@_error:n { bad~border }
7299 }

```

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).

```

7300 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7301 {
7302     \@@_qpoint:n \l_@@_tmpc_tl
7303     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7304     \@@_qpoint:n \l_tmpa_tl
7305     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7306     \@@_qpoint:n { #1 }
7307     \tl_if_empty:NTF \l_@@_borders_tikz_tl
7308     {
7309         \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7310         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7311         \pgfusepathqstroke
7312     }
7313     {
7314         \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7315         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7316     }
7317 }

```

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).

```

7318 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7319 {
7320     \@@_qpoint:n \l_@@_tmpd_tl
7321     \clist_if_in:NnTF \l_@@_borders_clist { left }
7322     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7323     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7324     \@@_qpoint:n \l_tmpb_tl

```

```

7325 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7326 \@@_qpoint:n { #1 }
7327 \tl_if_empty:NTF \l_@@_borders_tikz_tl
7328 {
7329   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7330   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7331   \pgfusepathqstroke
7332 }
7333 {
7334   \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7335   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7336 }
7337 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7338 \keys_define:nn { NiceMatrix / BlockBorders }
7339 {
7340   borders .clist_set:N = \l_@@_borders_clist ,
7341   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7342   rounded-corners .default:n = 4 pt ,
7343   line-width .dim_set:N = \l_@@_line_width_dim ,
7344 }

```

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.

```

7345 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7346 {
7347   \begin { tikzpicture }
7348   \clist_map_inline:nn { #5 }
7349   {
7350     \path [ ##1 ]
7351       ( #1 -| #2 )
7352       rectangle
7353       ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7354   }
7355   \end { tikzpicture }
7356 }

```

27 How to draw the dotted lines transparently

```

7357 \cs_set_protected:Npn \@@_renew_matrix:
7358 {
7359   \RenewDocumentEnvironment { pmatrix } { } {
7360     { \pNiceMatrix }
7361     { \endpNiceMatrix }
7362   \RenewDocumentEnvironment { vmatrix } { } {
7363     { \vNiceMatrix }
7364     { \endvNiceMatrix }
7365   \RenewDocumentEnvironment { Vmatrix } { } {
7366     { \VNiceMatrix }
7367     { \endVNiceMatrix }
7368   \RenewDocumentEnvironment { bmatrix } { } {
7369     { \bNiceMatrix }
7370     { \endbNiceMatrix }
7371   \RenewDocumentEnvironment { Bmatrix } { } {
7372     { \BNiceMatrix }
7373     { \endBNiceMatrix }
7374 }

```


28 Automatic arrays

We will extract the potential keys `columns-type`, `l`, `c`, `r` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7375 \keys_define:nn { NiceMatrix / Auto }
7376 {
7377   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7378   columns-type .value_required:n = true ,
7379   l .meta:n = { columns-type = l } ,
7380   r .meta:n = { columns-type = r } ,
7381   c .meta:n = { columns-type = c } ,
7382   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7383   delimiters / color .value_required:n = true ,
7384   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7385   delimiters / max-width .default:n = true ,
7386   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7387   delimiters .value_required:n = true ,
7388 }

7389 \NewDocumentCommand \AutoNiceMatrixWithDelims
7390 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7391 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

7392 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7393 {

```

The group is for the protection of the keys.

```

7394 \group_begin:
7395 \bool_set_true:N \l_@@_Matrix_bool
7396 \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`).

```

7397 \cs_set_eq:NN \@@_transform_preamble: \prg_do_nothing:
7398 \use:x
7399 {
7400   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7401   { * { #4 } { \exp_not:N \l_@@_columns_type_tl } }
7402   [ \exp_not:N \l_tmpa_tl ]
7403 }
7404 \int_compare:nNnT \l_@@_first_row_int = 0
7405 {
7406   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7407   \prg_replicate:nn { #4 - 1 } { & }
7408   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7409 }
7410 \prg_replicate:nn { #3 }
7411 {
7412   \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`).

```

7413   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7414   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7415 }
7416 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7417 {
7418   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7419   \prg_replicate:nn { #4 - 1 } { & }
7420   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7421 }
7422 \end { NiceArrayWithDelims }
7423 \group_end:

```

```

7424 }
7425 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7426 {
7427   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7428   {
7429     \bool_gset_false:N \g_@@_NiceArray_bool
7430     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7431     \AutoNiceMatrixWithDelims { #2 } { #3 }
7432   }
7433 }
7434 \@@_define_com:nnn p ( )
7435 \@@_define_com:nnn b [ ]
7436 \@@_define_com:nnn v | |
7437 \@@_define_com:nnn V \l \l
7438 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7439 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
7440 {
7441   \group_begin:
7442   \bool_gset_true:N \g_@@_NiceArray_bool
7443   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7444   \group_end:
7445 }

```

29 The redefinition of the command `\dotfill`

```

7446 \cs_set_eq:NN \@@_old_dotfill \dotfill
7447 \cs_new_protected:Npn \@@_dotfill:
7448 {

```

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}`).

```

7449   \@@_old_dotfill
7450   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7451 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

7452 \cs_new_protected:Npn \@@_dotfill_i:
7453 { \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.

```

7454 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7455 {
7456   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7457   {
7458     \@@_actually_diagbox:nnnnnn
7459     { \int_use:N \c_iRow }
7460     { \int_use:N \c_jCol }
7461     { \int_use:N \c_iRow }
7462     { \int_use:N \c_jCol }

```

```

7463         { \exp_not:n { #1 } }
7464         { \exp_not:n { #2 } }
7465     }

```

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`.

```

7466     \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7467     {
7468         { \int_use:N \c@iRow }
7469         { \int_use:N \c@jCol }
7470         { \int_use:N \c@iRow }
7471         { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7472         { }
7473     }
7474 }

```

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.

```

7475 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7476 {
7477     \pgfpicture
7478     \pgf@relevantforpicturesizefalse
7479     \pgfrememberpicturepositiononpagetrue
7480     \@@_qpoint:n { row - #1 }
7481     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7482     \@@_qpoint:n { col - #2 }
7483     \dim_set_eq:NN \l_tmpb_dim \pgf@x
7484     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7485     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7486     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7487     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7488     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7489     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7490     {

```

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.

```

7491     \CT@arc@
7492     \pgfsetroundcap
7493     \pgfusepathqstroke
7494 }
7495 \pgfset { inner~sep = 1 pt }
7496 \pgfscope
7497 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7498 \pgfnode { rectangle } { south~west }
7499 {
7500     \begin { minipage } { 20 cm }
7501     \@@_math_toggle_token: #5 \@@_math_toggle_token:
7502     \end { minipage }
7503 }
7504 { }
7505 { }
7506 \endpgfscope
7507 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7508 \pgfnode { rectangle } { north~east }
7509 {
7510     \begin { minipage } { 20 cm }
7511     \raggedleft
7512     \@@_math_toggle_token: #6 \@@_math_toggle_token:
7513     \end { minipage }

```

```

7514     }
7515     { }
7516     { }
7517 \endpgfpicture
7518 }

```

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.

```

7519 \keys_define:nn { NiceMatrix }
7520 {
7521   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7522   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7523 }
7524 \keys_define:nn { NiceMatrix / CodeAfter }
7525 {
7526   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7527   sub-matrix .value_required:n = true ,
7528   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7529   delimiters / color .value_required:n = true ,
7530   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7531   rules .value_required:n = true ,
7532   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
7533 }

```

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`.

```

7534 \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 `\`.

```

7535 \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`.

```

7536 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7537 {
7538   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7539   \@@_CodeAfter_iv:n
7540 }

```

We catch the argument of the command `\end` (in `#1`).

```

7541 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7542 {

```

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.

```

7543   \str_if_eq:eeTF \@currenvir { #1 }
7544   { \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`.

```

7545     {
7546         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7547         \@@_CodeAfter_ii:n
7548     }
7549 }
```

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.

```

7550 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7551 {
7552     \pgfpicture
7553     \pgfrememberpicturepositiononpagetrue
7554     \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.

```

7555     \@@_qpoint:n { row - 1 }
7556     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7557     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7558     \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).

```

7559     \bool_if:nTF { #3 }
7560     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7561     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7562     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7563     {
7564         \cs_if_exist:cT
7565         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7566         {
7567             \pgfpointanchor
7568             { \@@_env: - ##1 - #2 }
7569             { \bool_if:nTF { #3 } { west } { east } }
7570             \dim_set:Nn \l_tmpa_dim
7571             { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7572         }
7573     }
```

Now we can put the delimiter with a node of PGF.

```

7574     \pgfset { inner~sep = \c_zero_dim }
7575     \dim_zero:N \nulldelimiterspace
7576     \pgftransformshift
7577     {
7578         \pgfpoint
7579         { \l_tmpa_dim }
```

```

7580         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7581     }
7582     \pgfnode
7583     { rectangle }
7584     { \bool_if:nTF { #3 } { east } { west } }
7585     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7586         \nullfont
7587         \c_math_toggle_token
7588         \@@_color:V \l_@@_delimiters_color_tl
7589         \bool_if:nTF { #3 } { \left #1 } { \left . }
7590         \vcenter
7591         {
7592             \nullfont
7593             \hrule \@height
7594                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7595                 \@depth \c_zero_dim
7596                 \@width \c_zero_dim
7597         }
7598         \bool_if:nTF { #3 } { \right . } { \right #1 }
7599         \c_math_toggle_token
7600     }
7601     { }
7602     { }
7603     \endpgfpicture
7604 }

```

33 The command `\SubMatrix`

```

7605 \keys_define:nn { NiceMatrix / sub-matrix }
7606 {
7607     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7608     extra-height .value_required:n = true ,
7609     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7610     left-xshift .value_required:n = true ,
7611     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7612     right-xshift .value_required:n = true ,
7613     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7614     xshift .value_required:n = true ,
7615     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7616     delimiters / color .value_required:n = true ,
7617     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7618     slim .default:n = true ,
7619     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7620     hlines .default:n = all ,
7621     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7622     vlines .default:n = all ,
7623     hvlines .meta:n = { hlines, vlines } ,
7624     hvlines .value_forbidden:n = true ,
7625 }
7626 \keys_define:nn { NiceMatrix }
7627 {
7628     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7629     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7630     NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7631     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7632     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7633     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7634 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

7635 \keys_define:nn { NiceMatrix / SubMatrix }
7636 {
7637     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7638     delimiters / color .value_required:n = true ,
7639     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7640     hlines .default:n = all ,
7641     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7642     vlines .default:n = all ,
7643     hvlines .meta:n = { hlines, vlines } ,
7644     hvlines .value_forbidden:n = true ,
7645     name .code:n =
7646     \tl_if_empty:nTF { #1 }
7647     { \@@_error:n { Invalid-name } }
7648     {
7649         \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7650         {
7651             \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7652             { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7653             {
7654                 \str_set:Nn \l_@@_submatrix_name_str { #1 }
7655                 \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7656             }
7657         }
7658         { \@@_error:n { Invalid-name } }
7659     } ,
7660     name .value_required:n = true ,
7661     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7662     rules .value_required:n = true ,
7663     code .tl_set:N = \l_@@_code_tl ,
7664     code .value_required:n = true ,
7665     unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
7666 }

7667 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7668 {
7669     \peek_remove_spaces:n
7670     {
7671         \tl_gput_right:Nx \g_@@_pre_code_after_tl
7672         {
7673             \SubMatrix { #1 } { #2 } { #3 } { #4 }
7674             [
7675                 delimiters / color = \l_@@_delimiters_color_tl ,
7676                 hlines = \l_@@_submatrix_hlines_clist ,
7677                 vlines = \l_@@_submatrix_vlines_clist ,
7678                 extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
7679                 left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
7680                 right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
7681                 slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
7682                 #5
7683             ]
7684         }
7685         \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7686     }
7687 }

7688 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
7689 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7690 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

7691 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
7692 {
7693     \seq_gput_right:Nx \g_@@_submatrix_seq

```

```

7694     {
We use \str_if_eq:nnTF because it is fully expandable.
7695     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
7696     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
7697     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
7698     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
7699   }
7700 }

```

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.

```

7701 \hook_gput_code:nnn { begindocument } { . }
7702 {
7703   \tl_set:Nn \l_@@_argspec_tl { m m m 0 { } E { _ ^ } { { } { } } }
7704   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
7705   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
7706     {
7707       \peek_remove_spaces:n
7708       {
7709         \@@_sub_matrix:nnnnnnn
7710         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
7711       }
7712     }
7713 }

```

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).

```

7714 \NewDocumentCommand \@@_compute_i_j:nn
7715 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7716 { \@@_compute_i_j:nnnn #1 #2 }
7717 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
7718 {
7719   \tl_set:Nn \l_@@_first_i_tl { #1 }
7720   \tl_set:Nn \l_@@_first_j_tl { #2 }
7721   \tl_set:Nn \l_@@_last_i_tl { #3 }
7722   \tl_set:Nn \l_@@_last_j_tl { #4 }
7723   \tl_if_eq:NnT \l_@@_first_i_tl { last }
7724     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
7725   \tl_if_eq:NnT \l_@@_first_j_tl { last }
7726     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
7727   \tl_if_eq:NnT \l_@@_last_i_tl { last }
7728     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
7729   \tl_if_eq:NnT \l_@@_last_j_tl { last }
7730     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
7731 }

```



```

7732 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
7733 {
7734   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

7735   \@@_compute_i_j:nn { #2 } { #3 }
7736   \bool_lazy_or:nnTF
7737     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7738     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7739     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
7740     {
7741       \str_clear_new:N \l_@@_submatrix_name_str
7742       \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
7743       \pgfpicture
7744       \pgfrememberpicturepositiononpagetrue
7745       \pgf@relevantforpicturesizefalse
7746       \pgfset { inner~sep = \c_zero_dim }
7747       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7748       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

7749   \bool_if:NTF \l_@@_submatrix_slim_bool
7750     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
7751     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
7752     {
7753       \cs_if_exist:cT
7754         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7755         {
7756           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7757           \dim_set:Nn \l_@@_x_initial_dim
7758             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7759         }
7760       \cs_if_exist:cT
7761         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7762         {
7763           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7764           \dim_set:Nn \l_@@_x_final_dim
7765             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7766         }
7767     }
7768   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7769     { \@@_error:nn { Impossible-delimiter } { left } }
7770     {
7771       \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7772         { \@@_error:nn { Impossible-delimiter } { right } }
7773         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7774     }
7775   \endpgfpicture
7776 }
7777 \group_end:
7778 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7779 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7780 {
7781   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7782   \dim_set:Nn \l_@@_y_initial_dim
7783     {
7784       \fp_to_dim:n
7785         {
7786           \pgf@y
7787           + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
7788         }
7789     } % modified 6.13c

```

```

7790 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7791 \dim_set:Nn \l_@@_y_final_dim
7792 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
7793 % modified 6.13c
7794 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7795 {
7796   \cs_if_exist:cT
7797   { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7798   {
7799     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7800     \dim_set:Nn \l_@@_y_initial_dim
7801     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7802   }
7803   \cs_if_exist:cT
7804   { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7805   {
7806     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7807     \dim_set:Nn \l_@@_y_final_dim
7808     { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7809   }
7810 }
7811 \dim_set:Nn \l_tmpa_dim
7812 {
7813   \l_@@_y_initial_dim - \l_@@_y_final_dim +
7814   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7815 }
7816 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7817 \group_begin:
7818 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7819 \@@_set_CT@arc@:V \l_@@_rules_color_tl
7820 \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`.

```

7821 \seq_map_inline:Nn \g_@@_cols_vlism_seq
7822 {
7823   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7824   {
7825     \int_compare:nNnT
7826     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
7827     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

7828 \@@_qpoint:n { col - ##1 }
7829 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7830 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7831 \pgfusepathqstroke
7832 }
7833 }
7834 }

```

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.

```

7835 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7836 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7837 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7838 {
7839   \bool_lazy_and:nnTF
7840   { \int_compare_p:nNn { ##1 } > 0 }
7841   {

```

```

7842         \int_compare_p:nNn
7843         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7844     {
7845         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7846         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7847         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7848         \pgfusepathqstroke
7849     }
7850     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
7851 }

```

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.

```

7852 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7853 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7854 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist {
7855     {
7856         \bool_lazy_and:nnTF
7857         { \int_compare_p:nNn { ##1 } > 0 }
7858         {
7859             \int_compare_p:nNn
7860             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7861         {
7862             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7863 \group_begin:
7864 \dim_set:Nn \l_tmpa_dim
7865 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7866 \str_case:nn { #1 }
7867 {
7868     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7869     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7870     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7871 }
7872 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

7873 \dim_set:Nn \l_tmpb_dim
7874 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7875 \str_case:nn { #2 }
7876 {
7877     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7878     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7879     \{ { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7880 }
7881 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7882 \pgfusepathqstroke
7883 \group_end:
7884 }
7885 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7886 }

```

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).

```

7887 \str_if_empty:NF \l_@@_submatrix_name_str
7888 {
7889     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7890     \l_@@_x_initial_dim \l_@@_y_initial_dim
7891     \l_@@_x_final_dim \l_@@_y_final_dim
7892 }
7893 \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`.

```

7894 \begin { pgfscope }
7895 \pgftransformshift
7896 {
7897   \pgfpoint
7898   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7899   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7900 }
7901 \str_if_empty:NTF \l_@@_submatrix_name_str
7902 { \@@_node_left:nn #1 { } }
7903 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7904 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7905 \pgftransformshift
7906 {
7907   \pgfpoint
7908   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7909   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7910 }
7911 \str_if_empty:NTF \l_@@_submatrix_name_str
7912 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7913 {
7914   \@@_node_right:nnnn #2
7915   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7916 }
7917 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7918 \flag_clear_new:n { nicematrix }
7919 \l_@@_code_tl
7920 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-lj$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

7921 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument `#1` of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

7922 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
7923 {
7924   \use:e
7925   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
7926 }

```

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 -.

```

7927 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
7928 { #1 { \@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

7929 \tl_const:Nn \c_@@_integers_alist_tl
7930 {
7931   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7932   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7933   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7934   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7935 }

```

```

7936 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7937 {

```

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`.

```

7938   \tl_if_empty:nTF { #2 }
7939   {
7940     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7941     {
7942       \flag_raise:n { nicematrix }
7943       \int_if_even:nTF { \flag_height:n { nicematrix } }
7944       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7945       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7946     }
7947     { #1 }
7948   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

7949   { \@@_pgfpointanchor_iii:w { #1 } #2 }
7950 }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

7951 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
7952 {
7953   \str_case:nnF { #1 }
7954   {
7955     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7956     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7957   }

```

Now the case of a node of the form $i-j$.

```

7958   {
7959     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7960     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7961   }
7962 }

```

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`).

```

7963 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7964 {
7965   \pgfnode
7966   { rectangle }
7967   { east }
7968   {
7969     \nullfont

```

```

7970 \c_math_toggle_token
7971 \@@_color:V \l_@@_delimiters_color_tl
7972 \left #1
7973 \vcenter
7974 {
7975     \nullfont
7976     \hrule \@height \l_tmpa_dim
7977         \@depth \c_zero_dim
7978         \@width \c_zero_dim
7979 }
7980 \right .
7981 \c_math_toggle_token
7982 }
7983 { #2 }
7984 { }
7985 }

```

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.

```

7986 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
7987 {
7988     \pgfnode
7989     { rectangle }
7990     { west }
7991     {
7992         \nullfont
7993         \c_math_toggle_token
7994         \@@_color:V \l_@@_delimiters_color_tl
7995         \left .
7996         \vcenter
7997         {
7998             \nullfont
7999             \hrule \@height \l_tmpa_dim
8000                 \@depth \c_zero_dim
8001                 \@width \c_zero_dim
8002         }
8003         \right #1
8004         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8005         ^ { \smash { #4 } }
8006         \c_math_toggle_token
8007     }
8008     { #2 }
8009     { }
8010 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8011 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8012 {
8013     \peek_remove_spaces:n
8014     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8015 }
8016 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8017 {
8018     \peek_remove_spaces:n
8019     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8020 }

```

```

8021 \keys_define:nn { NiceMatrix / Brace }
8022 {
8023   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8024   left-shorten .default:n = true ,
8025   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8026   shorten .meta:n = { left-shorten , right-shorten } ,
8027   right-shorten .default:n = true ,
8028   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8029   yshift .value_required:n = true ,
8030   yshift .initial:n = \c_zero_dim ,
8031   color .tl_set:N = \l_tmpa_tl ,
8032   color .value_required:n = true ,
8033   unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
8034 }

```

#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*.

```

8035 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8036 {
8037   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8038   \@@_compute_i_j:nn { #1 } { #2 }
8039   \bool_lazy_or:nnTF
8040     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8041     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8042     {
8043       \str_if_eq:nnTF { #5 } { under }
8044       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8045       { \@@_error:nn { Construct~too~large } { \OverBrace } }
8046     }
8047     {
8048       \tl_clear:N \l_tmpa_tl
8049       \keys_set:nn { NiceMatrix / Brace } { #4 }
8050       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8051       \pgfpicture
8052       \pgfrememberpicturepositiononpagetrue
8053       \pgf@relevantforpicturesizefalse
8054       \bool_if:NT \l_@@_brace_left_shorten_bool
8055       {
8056         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8057         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8058         {
8059           \cs_if_exist:cT
8060             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8061             {
8062               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8063               \dim_set:Nn \l_@@_x_initial_dim
8064                 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8065             }
8066         }
8067       }
8068       \bool_lazy_or:nnT
8069       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8070       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8071       {
8072         \@@_qpoint:n { col - \l_@@_first_j_tl }
8073         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8074       }
8075       \bool_if:NT \l_@@_brace_right_shorten_bool
8076       {
8077         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8078         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl

```

```

8079         {
8080             \cs_if_exist:cT
8081             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8082             {
8083                 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8084                 \dim_set:Nn \l_@@_x_final_dim
8085                 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8086             }
8087         }
8088     }
8089     \bool_lazy_or:nnT
8090     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8091     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8092     {
8093         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8094         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8095     }
8096     \pgfset { inner~sep = \c_zero_dim }
8097     \str_if_eq:nnTF { #5 } { under }
8098     { \@@_underbrace_i:n { #3 } }
8099     { \@@_overbrace_i:n { #3 } }
8100     \endpgfpicture
8101 }
8102 \group_end:
8103 }

```

The argument is the text to put above the brace.

```

8104 \cs_new_protected:Npn \@@_overbrace_i:n #1
8105 {
8106     \@@_qpoint:n { row - \l_@@_first_i_tl }
8107     \pgftransformshift
8108     {
8109         \pgfpoint
8110         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8111         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8112     }
8113     \pgfnode
8114     { rectangle }
8115     { south }
8116     {
8117         \vbox_top:n
8118         {
8119             \group_begin:
8120             \everycr { }
8121             \halign
8122             {
8123                 \hfil ## \hfil \crrc
8124                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8125                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8126                 \c_math_toggle_token
8127                 \overbrace
8128                 {
8129                     \hbox_to_wd:nn
8130                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8131                     { }
8132                 }
8133                 \c_math_toggle_token
8134                 \cr
8135             }
8136             \group_end:
8137         }
8138     }
8139     { }
8140     { }

```



```
8141 }
```

The argument is the text to put under the brace.

```
8142 \cs_new_protected:Npn \@@_underbrace_i:n #1
8143 {
8144   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8145   \pgftransformshift
8146   {
8147     \pgfpoint
8148     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8149     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8150   }
8151   \pgfnode
8152   { rectangle }
8153   { north }
8154   {
8155     \group_begin:
8156     \everycr { }
8157     \vbox:n
8158     {
8159       \halign
8160       {
8161         \hfil ## \hfil \crrc
8162         \c_math_toggle_token
8163         \underbrace
8164         {
8165           \hbox_to_wd:nn
8166           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8167           { }
8168         }
8169         \c_math_toggle_token
8170         \cr
8171         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8172         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8173       }
8174     }
8175     \group_end:
8176   }
8177   { }
8178   { }
8179 }
```

35 The command \ShowCellNames

```
8180 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8181 {
8182   \dim_zero_new:N \g_@@_tmpc_dim
8183   \dim_zero_new:N \g_@@_tmpd_dim
8184   \dim_zero_new:N \g_@@_tmpe_dim
8185   \int_step_inline:nn \c@iRow
8186   {
8187     \begin { pgfpicture }
8188     \@@_qpoint:n { row - ##1 }
8189     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8190     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8191     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8192     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8193     \bool_if:NTF \l_@@_in_code_after_bool
8194     \end { pgfpicture }
```

```

8195 \int_step_inline:nn \c@jCol
8196 {
8197     \hbox_set:Nn \l_tmpa_box
8198     { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
8199     \begin { pgfpicture }
8200     \@@_qpoint:n { col - ####1 }
8201     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8202     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8203     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8204     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8205     \endpgfpicture
8206     \end { pgfpicture }
8207     \fp_set:Nn \l_tmpa_fp
8208     {
8209         \fp_min:nn
8210         {
8211             \fp_min:nn
8212             { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8213             { \dim_ratio:nn { \g_@@_tmpe_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8214         }
8215         { 1.0 }
8216     }
8217     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8218     \pgfpicture
8219     \pgfrememberpicturepositiononpagetrue
8220     \pgf@relevantforpicturesizefalse
8221     \pgftransformshift
8222     {
8223         \pgfpoint
8224         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8225         { \dim_use:N \g_@@_tmpa_dim }
8226     }
8227     \pgfnode
8228     { rectangle }
8229     { center }
8230     { \box_use:N \l_tmpa_box }
8231     { }
8232     { }
8233     \endpgfpicture
8234 }
8235 }
8236 }
8237 \NewDocumentCommand \@@_ShowCellNames { }
8238 {
8239     \bool_if:NT \l_@@_in_code_after_bool
8240     {
8241         \pgfpicture
8242         \pgfrememberpicturepositiononpagetrue
8243         \pgf@relevantforpicturesizefalse
8244         \pgfpathrectanglecorners
8245         { \@@_qpoint:n { 1 } }
8246         {
8247             \@@_qpoint:n
8248             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8249         }
8250         \pgfsetfillopacity { 0.75 }
8251         \pgfsetfillcolor { white }
8252         \pgfusepathqfill
8253         \endpgfpicture
8254     }
8255     \dim_zero_new:N \g_@@_tmpc_dim
8256     \dim_zero_new:N \g_@@_tmpd_dim
8257     \dim_zero_new:N \g_@@_tmpe_dim

```

```

8258 \int_step_inline:nn \c@iRow
8259 {
8260   \bool_if:NTF \l_@@_in_code_after_bool
8261   {
8262     \pgfpicture
8263     \pgfrememberpicturepositiononpagetrue
8264     \pgf@relevantforpicturesizefalse
8265   }
8266   { \begin { pgfpicture } }
8267   \@@_qpoint:n { row - ##1 }
8268   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8269   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8270   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8271   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8272   \bool_if:NTF \l_@@_in_code_after_bool
8273   { \endpgfpicture }
8274   { \end { pgfpicture } }
8275   \int_step_inline:nn \c@jCol
8276   {
8277     \hbox_set:Nn \l_tmpa_box
8278     {
8279       \normalfont \Large \sffamily \bfseries
8280       \bool_if:NTF \l_@@_in_code_after_bool
8281       { \color { red } }
8282       { \color { red ! 50 } }
8283       ##1 - #####1
8284     }
8285     \bool_if:NTF \l_@@_in_code_after_bool
8286     {
8287       \pgfpicture
8288       \pgfrememberpicturepositiononpagetrue
8289       \pgf@relevantforpicturesizefalse
8290     }
8291     { \begin { pgfpicture } }
8292     \@@_qpoint:n { col - #####1 }
8293     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8294     \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8295     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8296     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8297     \bool_if:NTF \l_@@_in_code_after_bool
8298     { \endpgfpicture }
8299     { \end { pgfpicture } }
8300     \fp_set:Nn \l_tmpa_fp
8301     {
8302       \fp_min:nn
8303       {
8304         \fp_min:nn
8305         { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8306         { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8307       }
8308       { 1.0 }
8309     }
8310     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8311     \pgfpicture
8312     \pgfrememberpicturepositiononpagetrue
8313     \pgf@relevantforpicturesizefalse
8314     \pgftransformshift
8315     {
8316       \pgfpoint
8317       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8318       { \dim_use:N \g_tmpa_dim }
8319     }
8320     \pgfnode

```

```

8321         { rectangle }
8322         { center }
8323         { \box_use:N \l_tmpa_box }
8324         { }
8325         { }
8326     \endpgfpicture
8327 }
8328 }
8329 }

```

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.

```

8330 \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.

```

8331 \bool_new:N \c_@@_footnote_bool
8332 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
8333 {
8334     The~key~'\l_keys_key_str'~is~unknown.  \
8335     That~key~will~be~ignored.  \
8336     For~a~list~of~the~available~keys,~type~H~<return>.
8337 }
8338 {
8339     The~available~keys~are~(in~alphabetic~order):~
8340     footnote,~
8341     footnotehyper,~
8342     messages-for-Overleaf,~
8343     no-test-for-array,~
8344     renew-dots,~and
8345     renew-matrix.
8346 }
8347 \keys_define:nn { NiceMatrix / Package }
8348 {
8349     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8350     renew-dots .value_forbidden:n = true ,
8351     renew-matrix .code:n = \@@_renew_matrix: ,
8352     renew-matrix .value_forbidden:n = true ,
8353     messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
8354     footnote .bool_set:N = \c_@@_footnote_bool ,
8355     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
8356     no-test-for-array .bool_set:N = \c_@@_no_test_for_array_bool ,
8357     no-test-for-array .default:n = true ,
8358     unknown .code:n = \@@_error:n { Unknown~key~for~package }
8359 }
8360 \ProcessKeysOptions { NiceMatrix / Package }

8361 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8362 {
8363     You~can't~use~the~option~'footnote'~because~the~package~
8364     footnotehyper~has~already~been~loaded.~
8365     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~

```

```

8366   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8367   of~the~package~footnotehyper.\\
8368   The~package~footnote~won't~be~loaded.
8369 }

8370 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8371 {
8372   You~can't~use~the~option~'footnotehyper'~because~the~package~
8373   footnote~has~already~been~loaded.~
8374   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8375   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8376   of~the~package~footnote.\\
8377   The~package~footnotehyper~won't~be~loaded.
8378 }

```

```

8379 \bool_if:NT \c_@@_footnote_bool
8380 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8381   \IfClassLoadTF { beamer }
8382   { \bool_set_false:N \c_@@_footnote_bool }
8383   {
8384     \IfPackageLoadedTF { footnotehyper }
8385     { \@@_error:n { footnote~with~footnotehyper~package } }
8386     { \usepackage { footnote } }
8387   }
8388 }

8389 \bool_if:NT \c_@@_footnotehyper_bool
8390 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8391   \IfClassLoadedTF { beamer }
8392   { \bool_set_false:N \c_@@_footnote_bool }
8393   {
8394     \IfPackageLoadedTF { footnote }
8395     { \@@_error:n { footnotehyper~with~footnote~package } }
8396     { \usepackage { footnotehyper } }
8397   }
8398   \bool_set_true:N \c_@@_footnote_bool
8399 }

```

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.

```

8400 \bool_new:N \l_@@_underscore_loaded_bool
8401 \IfPackageLoadedTF { underscore }
8402 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8403 { }

8404 \hook_gput_code:nnn { begindocument } { . }
8405 {
8406   \bool_if:NF \l_@@_underscore_loaded_bool
8407   {
8408     \IfPackageLoadedTF { underscore }

```

```

8409         { \@@_error:n { underscore~after~nicematrix } }
8410     { }
8411 }
8412 }

```

38 Error messages of the package

```

8413 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
8414 { \str_const:Nn \c_@@_available_keys_str { } }
8415 {
8416     \str_const:Nn \c_@@_available_keys_str
8417     { For~a~list~of~the~available~keys,~type~H~<return>. }
8418 }
8419 \seq_new:N \g_@@_types_of_matrix_seq
8420 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8421 {
8422     NiceMatrix ,
8423     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8424 }
8425 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8426 { \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`.

```

8427 \cs_new_protected:Npn \@@_error_too_much_cols:
8428 {
8429     \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8430     {
8431         \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8432         { \@@_fatal:n { too~much~cols~for~matrix } }
8433         {
8434             \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8435             { \@@_fatal:n { too~much~cols~for~matrix } }
8436             {
8437                 \bool_if:NF \l_@@_last_col_without_value_bool
8438                 { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
8439             }
8440         }
8441     }
8442     {
8443         \IfPackageLoadedTF { tabularx }
8444         {
8445             \str_if_eq:VnTF \g_@@_name_env_str { NiceTabularX }
8446             {
8447                 \int_compare:nNnTF \c@iRow = \c_zero_int
8448                 { \@@_fatal:n { X~columns~with~tabularx } }
8449                 {
8450                     \@@_fatal:nn { too~much~cols~for~array }
8451                     {
8452                         However,~this~message~may~be~erroneous:~
8453                         maybe~you~have~used~X~columns~while~'tabularx'~is~loaded,~
8454                         ~which~is~forbidden~(however,~it's~still~possible~to~use~
8455                         X~columns~in~{NiceTabularX}).
8456                     }
8457                 }
8458             }
8459         }
8460     }

```

```

8459         { \@@_fatal:nn { too-much-cols-for-array } { } }
8460     }
8461     { \@@_fatal:nn { too-much-cols-for-array } { } }
8462 }
8463 }

```

The following command must *not* be protected since it's used in an error message.

```

8464 \cs_new:Npn \@@_message_hdotsfor:
8465 {
8466     \tl_if_empty:VF \g_@@_HVDotsfor_lines_tl
8467     { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
8468 }
8469 \@@_msg_new:nn { negative-weight }
8470 {
8471     Negative-weight.\
8472     The-weight-of-the-'X'~columns-must-be-positive-and-you-have-used~
8473     the-value-\int_use:N \l_@@_weight_int'.\
8474     The-absolute-value-will-be-used.
8475 }
8476 \@@_msg_new:nn { last-col-not-used }
8477 {
8478     Column-not-used.\
8479     The-key-'last-col'-is-in-force-but-you-have-not-used-that-last-column~
8480     in-your-\@@_full_name_env:~However,~you-can-go-on.
8481 }
8482 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
8483 {
8484     Too-much-columns.\
8485     In-the-row-\int_eval:n { \c@iRow },~
8486     you-try-to-use-more-columns~
8487     than-allowed-by-your-\@@_full_name_env:.\@@_message_hdotsfor:\
8488     The-maximal-number-of-columns-is-\int_eval:n { \l_@@_last_col_int - 1 }~
8489     (plus-the-exterior-columns).~This-error-is-fatal.
8490 }
8491 \@@_msg_new:nn { too-much-cols-for-matrix }
8492 {
8493     Too-much-columns.\
8494     In-the-row-\int_eval:n { \c@iRow },~
8495     you-try-to-use-more-columns-than-allowed-by-your~
8496     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
8497     number-of-columns-for-a-matrix~(excepted-the-potential-exterior~
8498     columns)~is-fixed-by-the-LaTeX-counter-'MaxMatrixCols'.~
8499     Its-current-value-is-\int_use:N \c@MaxMatrixCols\ (use~
8500     \token_to_str:N \setcounter\ to-change-that-value).~
8501     This-error-is-fatal.
8502 }
8503 \@@_msg_new:nn { too-much-cols-for-array }
8504 {
8505     Too-much-columns.\
8506     In-the-row-\int_eval:n { \c@iRow },~
8507     ~you-try-to-use-more-columns-than-allowed-by-your~
8508     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
8509     \int_use:N \g_@@_static_num_of_col_int\
8510     ~(plus-the-potential-exterior-ones).~#1
8511     This-error-is-fatal.
8512 }
8513 \@@_msg_new:nn { X-columns-with-tabularx }
8514 {
8515     There-is-a-problem.\
8516     You-have-probably-used-X-columns-in-your-environment~{\g_@@_name_env_str}.~
8517     That's-not-allowed-because-'tabularx'-is-loaded~(however,~you-can-use-X-columns~

```

```

8518     in~an~environment~{NiceTabularX}).\\
8519     This~error~is~fatal.
8520 }

8521 \@@_msg_new:nn { columns~not~used }
8522 {
8523     Columns~not~used.\\
8524     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8525     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
8526     The~columns~you~did~not~used~won't~be~created.\\
8527     We~won't~have~similar~error~till~the~end~of~the~document.
8528 }

8529 \@@_msg_new:nn { in~first~col }
8530 {
8531     Erroneous~use.\\
8532     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
8533     That~command~will~be~ignored.
8534 }

8535 \@@_msg_new:nn { in~last~col }
8536 {
8537     Erroneous~use.\\
8538     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
8539     That~command~will~be~ignored.
8540 }

8541 \@@_msg_new:nn { in~first~row }
8542 {
8543     Erroneous~use.\\
8544     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
8545     That~command~will~be~ignored.
8546 }

8547 \@@_msg_new:nn { in~last~row }
8548 {
8549     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
8550     That~command~will~be~ignored.
8551 }

8552 \@@_msg_new:nn { caption~outside~float }
8553 {
8554     Key~caption~forbidden.\\
8555     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
8556     environment.~This~key~will~be~ignored.
8557 }

8558 \@@_msg_new:nn { short~caption~without~caption }
8559 {
8560     You~should~not~use~the~key~'short~caption'~without~'caption'.~
8561     However,~your~'short~caption'~will~be~used~as~'caption'.
8562 }

8563 \@@_msg_new:nn { double~closing~delimiter }
8564 {
8565     Double~delimiter.\\
8566     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
8567     delimiter.~This~delimiter~will~be~ignored.
8568 }

8569 \@@_msg_new:nn { delimiter~after~opening }
8570 {
8571     Double~delimiter.\\
8572     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
8573     delimiter.~That~delimiter~will~be~ignored.
8574 }

8575 \@@_msg_new:nn { bad~option~for~line~style }
8576 {

```



```

8577     Bad~line~style.\\
8578     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
8579     is~'standard'.~That~key~will~be~ignored.
8580 }

8581 \@@_msg_new:nn { Identical~notes~in~caption }
8582 {
8583     Identical~tabular~notes.\\
8584     You~can't~put~several~notes~with~the~same~content~in~
8585     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
8586     If~you~go~on,~the~output~will~probably~be~erroneous.
8587 }

8588 \@@_msg_new:nn { tabularnote~below~the~tabular }
8589 {
8590     \token_to_str:N \tabularnote\ forbidden\\
8591     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8592     of~your~tabular~because~the~caption~will~be~composed~below~
8593     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8594     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
8595     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8596     no~similar~error~will~raised~in~this~document.
8597 }

8598 \@@_msg_new:nn { Unknown~key~for~rules }
8599 {
8600     Unknown~key.\\
8601     There~is~only~two~keys~available~here:~width~and~color.\\
8602     You~key~'\l_keys_key_str'~will~be~ignored.
8603 }

8604 \@@_msg_new:nnn { Unknown~key~for~custom~line }
8605 {
8606     Unknown~key.\\
8607     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
8608     It~you~go~on,~you~will~probably~have~other~errors. \\
8609     \c_@@_available_keys_str
8610 }
8611 {
8612     The~available~keys~are~(in~alphabetic~order):~
8613     ccommand,~
8614     color,~
8615     command,~
8616     dotted,~
8617     letter,~
8618     multiplicity,~
8619     sep-color,~
8620     tikz,~and~total-width.
8621 }

8622 \@@_msg_new:nnn { Unknown~key~for~xdots }
8623 {
8624     Unknown~key.\\
8625     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
8626     \c_@@_available_keys_str
8627 }
8628 {
8629     The~available~keys~are~(in~alphabetic~order):~
8630     'color',~
8631     'inter',~
8632     'line-style',~
8633     'radius',~
8634     'shorten',~
8635     'shorten-end'~and~'shorten-start'.
8636 }

8637 \@@_msg_new:nn { Unknown~key~for~rowcolors }

```

```

8638 {
8639   Unknown~key.\\
8640   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
8641   (and~you~try~to~use~'\l_keys_key_str')\\
8642   That~key~will~be~ignored.
8643 }
8644 \@@_msg_new:nn { label~without~caption }
8645 {
8646   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
8647   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
8648 }
8649 \@@_msg_new:nn { W~warning }
8650 {
8651   Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
8652   (row~\int_use:N \c@iRow).
8653 }
8654 \@@_msg_new:nn { Construct~too~large }
8655 {
8656   Construct~too~large.\\
8657   Your~command~\token_to_str:N #1
8658   can't~be~drawn~because~your~matrix~is~too~small.\\
8659   That~command~will~be~ignored.
8660 }
8661 \@@_msg_new:nn { underscore~after~nicematrix }
8662 {
8663   Problem~with~'underscore'.\\
8664   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
8665   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
8666   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
8667 }
8668 \@@_msg_new:nn { ampersand~in~light-syntax }
8669 {
8670   Ampersand~forbidden.\\
8671   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
8672   the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
8673 }
8674 \@@_msg_new:nn { double~backslash~in~light-syntax }
8675 {
8676   Double~backslash~forbidden.\\
8677   You~can't~use~\token_to_str:N
8678   \\~to~separate~rows~because~the~key~'light-syntax'~
8679   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
8680   (set~by~the~key~'end-of-row').~This~error~is~fatal.
8681 }
8682 \@@_msg_new:nn { hlines~with~color }
8683 {
8684   Incompatible~keys.\\
8685   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
8686   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
8687   Maybe~it~will~possible~in~future~version.\\
8688   Your~key~will~be~discarded.
8689 }
8690 \@@_msg_new:nn { bad~value~for~baseline }
8691 {
8692   Bad~value~for~baseline.\\
8693   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
8694   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
8695   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
8696   the~form~'line-i'.\\
8697   A~value~of~1~will~be~used.

```

```

8698     }
8699 \@@_msg_new:nn { ragged2e~not~loaded }
8700 {
8701     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
8702     your~column~'\l_@@_vpos_col_str'~(or~'X')~.~The~key~'\str_lowercase:V
8703     \l_keys_key_str'~will~be~used~instead.
8704 }
8705 \@@_msg_new:nn { Invalid~name }
8706 {
8707     Invalid~name.\\
8708     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
8709     \SubMatrix\ of~your~\@@_full_name_env:.\\
8710     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
8711     This~key~will~be~ignored.
8712 }
8713 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
8714 {
8715     Wrong~line.\\
8716     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
8717     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
8718     number~is~not~valid.~It~will~be~ignored.
8719 }
8720 \@@_msg_new:nn { Impossible~delimiter }
8721 {
8722     Impossible~delimiter.\\
8723     It's~impossible~to~draw~the~#1~delimiter~of~your~
8724     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
8725     in~that~column.
8726     \bool_if:NT \l_@@_submatrix_slim_bool
8727     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
8728     This~\token_to_str:N \SubMatrix\ will~be~ignored.
8729 }
8730 \@@_msg_new:nn { width~without~X~columns }
8731 {
8732     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
8733     That~key~will~be~ignored.
8734 }
8735 \@@_msg_new:nn { key~multiplicity~with~dotted }
8736 {
8737     Incompatible~keys. \\
8738     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
8739     in~a~'custom-line'.~They~are~incompatible. \\
8740     The~key~'multiplicity'~will~be~discarded.
8741 }
8742 \@@_msg_new:nn { empty~environment }
8743 {
8744     Empty~environment.\\
8745     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
8746 }
8747 \@@_msg_new:nn { No~letter~and~no~command }
8748 {
8749     Erroneous~use.\\
8750     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
8751     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
8752     '~ccommand'~(to~draw~horizontal~rules).\\
8753     However,~you~can~go~on.
8754 }
8755 \@@_msg_new:nn { Forbidden~letter }
8756 {
8757     Forbidden~letter.\\

```

```

8758     You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
8759     It~will~be~ignored.
8760 }

8761 \@@_msg_new:nn { Several~letters }
8762 {
8763     Wrong~name.\\
8764     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
8765     have~used~'\l_@@_letter_str').\\
8766     It~will~be~ignored.
8767 }

8768 \@@_msg_new:nn { Delimiter~with~small }
8769 {
8770     Delimiter~forbidden.\\
8771     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
8772     because~the~key~'small'~is~in~force.\\
8773     This~error~is~fatal.
8774 }

8775 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
8776 {
8777     Unknown~cell.\\
8778     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
8779     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
8780     can't~be~executed~because~a~cell~doesn't~exist.\\
8781     This~command~\token_to_str:N \line\ will~be~ignored.
8782 }

8783 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
8784 {
8785     Duplicate~name.\\
8786     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
8787     in~this~\@@_full_name_env:..\\
8788     This~key~will~be~ignored.\\
8789     \bool_if:NF \c_@@_messages_for_Overleaf_bool
8790     { For~a~list~of~the~names~already~used,~type~H~<return>. }
8791 }
8792 {
8793     The~names~already~defined~in~this~\@@_full_name_env:~ are:~
8794     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
8795 }

8796 \@@_msg_new:nn { r~or~l~with~preamble }
8797 {
8798     Erroneous~use.\\
8799     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:..~
8800     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
8801     your~\@@_full_name_env:..\\
8802     This~key~will~be~ignored.
8803 }

8804 \@@_msg_new:nn { Hdotsfor~in~col~0 }
8805 {
8806     Erroneous~use.\\
8807     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
8808     the~array.~This~error~is~fatal.
8809 }

8810 \@@_msg_new:nn { bad~corner }
8811 {
8812     Bad~corner.\\
8813     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
8814     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
8815     This~specification~of~corner~will~be~ignored.
8816 }

8817 \@@_msg_new:nn { bad~border }

```

```

8818 {
8819   Bad~border.\\
8820   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
8821   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
8822   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
8823   also~use~the~key~'tikz'
8824   \IfPackageLoadedTF { tikz }
8825     { }
8826     {~if~you~load~the~LaTeX~package~'tikz'})\\.\\
8827   This~specification~of~border~will~be~ignored.
8828 }
8829 \@@_msg_new:nn { tikz~key~without~tikz }
8830 {
8831   Tikz~not~loaded.\\
8832   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
8833   \Block'~because~you~have~not~loaded~tikz.~
8834   This~key~will~be~ignored.
8835 }
8836 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
8837 {
8838   Erroneous~use.\\
8839   In~the~\@@_full_name_env:,~you~must~use~the~key~
8840   'last-col'~without~value.\\
8841   However,~you~can~go~on~for~this~time~
8842   (the~value~'\l_keys_value_tl'~will~be~ignored).
8843 }
8844 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
8845 {
8846   Erroneous~use.\\
8847   In~\NiceMatrixoptions,~you~must~use~the~key~
8848   'last-col'~without~value.\\
8849   However,~you~can~go~on~for~this~time~
8850   (the~value~'\l_keys_value_tl'~will~be~ignored).
8851 }
8852 \@@_msg_new:nn { Block~too~large~1 }
8853 {
8854   Block~too~large.\\
8855   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
8856   too~small~for~that~block. \\
8857 }
8858 \@@_msg_new:nn { Block~too~large~2 }
8859 {
8860   Block~too~large.\\
8861   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8862   \g_@@_static_num_of_col_int\
8863   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
8864   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
8865   (&)~at~the~end~of~the~first~row~of~your~
8866   \@@_full_name_env:.\\
8867   This~block~and~maybe~others~will~be~ignored.
8868 }
8869 \@@_msg_new:nn { unknown~column~type }
8870 {
8871   Bad~column~type.\\
8872   The~column~type~'#1'~in~your~\@@_full_name_env:\
8873   is~unknown. \\
8874   This~error~is~fatal.
8875 }
8876 \@@_msg_new:nn { unknown~column~type~S }
8877 {
8878   Bad~column~type.\\

```

```

8879 The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
8880 If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
8881 load~that~package. \\
8882 This~error~is~fatal.
8883 }

8884 \@@_msg_new:nn { tabularnote~forbidden }
8885 {
8886   Forbidden~command.\\
8887   You~can't~use~the~command~\token_to_str:N\tabularnote\
8888   ~here.~This~command~is~available~only~in~
8889   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
8890   the~argument~of~a~command~\token_to_str:N \caption\ included~
8891   in~an~environment~{table}. \\
8892   This~command~will~be~ignored.
8893 }

8894 \@@_msg_new:nn { borders~forbidden }
8895 {
8896   Forbidden~key.\\
8897   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
8898   because~the~option~'rounded~corners'~
8899   is~in~force~with~a~non~zero~value.\\
8900   This~key~will~be~ignored.
8901 }

8902 \@@_msg_new:nn { bottomrule~without~booktabs }
8903 {
8904   booktabs~not~loaded.\\
8905   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
8906   loaded~'booktabs'.\\
8907   This~key~will~be~ignored.
8908 }

8909 \@@_msg_new:nn { enumitem~not~loaded }
8910 {
8911   enumitem~not~loaded.\\
8912   You~can't~use~the~command~\token_to_str:N\tabularnote\
8913   ~because~you~haven't~loaded~'enumitem'.\\
8914   All~the~commands~\token_to_str:N\tabularnote\ will~be~
8915   ignored~in~the~document.
8916 }

8917 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
8918 {
8919   Tikz~not~loaded.\\
8920   You~have~used~the~key~'tikz'~in~the~definition~of~a~
8921   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
8922   You~can~go~on~but~you~will~have~another~error~if~you~actually~
8923   use~that~custom~line.
8924 }

8925 \@@_msg_new:nn { tikz~in~borders~without~tikz }
8926 {
8927   Tikz~not~loaded.\\
8928   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
8929   command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
8930   That~key~will~be~ignored.
8931 }

8932 \@@_msg_new:nn { color~in~custom~line~with~tikz }
8933 {
8934   Erroneous~use.\\
8935   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
8936   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
8937   The~key~'color'~will~be~discarded.
8938 }

```

```

8939 \@@_msg_new:nn { Wrong-last-row }
8940 {
8941   Wrong-number.\
8942   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
8943   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
8944   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
8945   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
8946   without~value~(more~compilations~might~be~necessary).
8947 }
8948 \@@_msg_new:nn { Yet~in~env }
8949 {
8950   Nested~environments.\
8951   Environments~of~nicematrix~can't~be~nested.\
8952   This~error~is~fatal.
8953 }
8954 \@@_msg_new:nn { Outside-math-mode }
8955 {
8956   Outside~math~mode.\
8957   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
8958   (and~not~in~\token_to_str:N \vcenter).\
8959   This~error~is~fatal.
8960 }
8961 \@@_msg_new:nn { One~letter~allowed }
8962 {
8963   Bad~name.\
8964   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
8965   It~will~be~ignored.
8966 }
8967 \@@_msg_new:nn { TabularNote~in~CodeAfter }
8968 {
8969   Environment~{TabularNote}~forbidden.\
8970   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
8971   but~*before*~the~\token_to_str:N \CodeAfter.\
8972   This~environment~{TabularNote}~will~be~ignored.
8973 }
8974 \@@_msg_new:nn { varwidth~not~loaded }
8975 {
8976   varwidth~not~loaded.\
8977   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
8978   loaded.\
8979   Your~column~will~behave~like~'p'.
8980 }
8981 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
8982 {
8983   Unknow~key.\
8984   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
8985   \c_@@_available_keys_str
8986 }
8987 {
8988   The~available~keys~are~(in~alphabetic~order):~
8989   color,~
8990   dotted,~
8991   multiplicity,~
8992   sep-color,~
8993   tikz,~and~total~width.
8994 }
8995
8996 \@@_msg_new:nnn { Unknown~key~for~Block }
8997 {
8998   Unknown~key.\
8999   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N

```

```

9000 \Block.\ It~will~be~ignored. \
9001 \c_@@_available_keys_str
9002 }
9003 {
9004 The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9005 hlines,~hvlines,~l,~line~width,~name,~rounded~corners,~r,~respect~arraystretch,~
9006 t,~T,~tikz,~transparent~and~vlines.
9007 }
9008 \@@_msg_new:nn { Version~of~siunitx~too~old }
9009 {
9010 siunitx~too~old.\
9011 You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9012 is~too~old.~You~need~at~least~v~3.0.38~and~your~log~file~says:~"siunitx,~
9013 \use:c { ver @ siunitx.sty }". \
9014 This~error~is~fatal.
9015 }
9016 \@@_msg_new:nnn { Unknown~key~for~Brace }
9017 {
9018 Unknown~key.\
9019 The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9020 \UnderBrace\ and~\token_to_str:N \OverBrace.\
9021 It~will~be~ignored. \
9022 \c_@@_available_keys_str
9023 }
9024 {
9025 The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9026 right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9027 right~shorten)~and~yshift.
9028 }
9029 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9030 {
9031 Unknown~key.\
9032 The~key~'\l_keys_key_str'~is~unknown.\
9033 It~will~be~ignored. \
9034 \c_@@_available_keys_str
9035 }
9036 {
9037 The~available~keys~are~(in~alphabetic~order):~
9038 delimiters/color,~
9039 rules~(with~the~subkeys~'color'~and~'width'),~
9040 sub~matrix~(several~subkeys)~
9041 and~xdots~(several~subkeys).~
9042 The~latter~is~for~the~command~\token_to_str:N \line.
9043 }
9044 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9045 {
9046 Unknown~key.\
9047 The~key~'\l_keys_key_str'~is~unknown.\
9048 It~will~be~ignored. \
9049 \c_@@_available_keys_str
9050 }
9051 {
9052 The~available~keys~are~(in~alphabetic~order):~
9053 create~cell~nodes,~
9054 delimiters/color~and~
9055 sub~matrix~(several~subkeys).
9056 }
9057 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9058 {
9059 Unknown~key.\
9060 The~key~'\l_keys_key_str'~is~unknown.\
9061 That~key~will~be~ignored. \

```



```

9062     \c_@@_available_keys_str
9063 }
9064 {
9065     The~available~keys~are~(in~alphabetic~order):~
9066     'delimiters/color',~
9067     'extra-height',~
9068     'hlines',~
9069     'hvlines',~
9070     'left-xshift',~
9071     'name',~
9072     'right-xshift',~
9073     'rules'~(with~the~subkeys~'color'~and~'width'),~
9074     'slim',~
9075     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9076     and~'right-xshift').\\
9077 }
9078 \@@_msg_new:nnn { Unknown~key~for~notes }
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     bottomrule,~
9088     code-after,~
9089     code-before,~
9090     detect-duplicates,~
9091     enumitem-keys,~
9092     enumitem-keys-para,~
9093     para,~
9094     label-in-list,~
9095     label-in-tabular~and~
9096     style.
9097 }
9098 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9099 {
9100     Unknown~key.\\
9101     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9102     \token_to_str:N \RowStyle. \\
9103     That~key~will~be~ignored. \\
9104     \c_@@_available_keys_str
9105 }
9106 {
9107     The~available~keys~are~(in~alphabetic~order):~
9108     'bold',~
9109     'cell-space-top-limit',~
9110     'cell-space-bottom-limit',~
9111     'cell-space-limits',~
9112     'color',~
9113     'nb-rows'~and~
9114     'rowcolor'.
9115 }
9116 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9117 {
9118     Unknown~key.\\
9119     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9120     \token_to_str:N \NiceMatrixOptions. \\
9121     That~key~will~be~ignored. \\
9122     \c_@@_available_keys_str
9123 }
9124 {

```

```

9125 The~available~keys~are~(in~alphabetic~order):~
9126 allow~duplicate~names,~
9127 caption~above,~
9128 cell~space~bottom~limit,~
9129 cell~space~limits,~
9130 cell~space~top~limit,~
9131 code~for~first~col,~
9132 code~for~first~row,~
9133 code~for~last~col,~
9134 code~for~last~row,~
9135 corners,~
9136 custom~key,~
9137 create~extra~nodes,~
9138 create~medium~nodes,~
9139 create~large~nodes,~
9140 delimiters~(several~subkeys),~
9141 end~of~row,~
9142 first~col,~
9143 first~row,~
9144 hlines,~
9145 hvlines,~
9146 hvlines~except~borders,~
9147 last~col,~
9148 last~row,~
9149 left~margin,~
9150 light~syntax,~
9151 matrix/columns~type,~
9152 notes~(several~subkeys),~
9153 nullify~dots,~
9154 pgf~node~code,~
9155 renew~dots,~
9156 renew~matrix,~
9157 respect~arraystretch,~
9158 right~margin,~
9159 rules~(with~the~subkeys~'color'~and~'width'),~
9160 small,~
9161 sub~matrix~(several~subkeys),~
9162 vlines,~
9163 xdots~(several~subkeys).
9164 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9165 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9166 {
9167   Unknown~key.\\
9168   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9169   \{NiceArray\}. \\
9170   That~key~will~be~ignored. \\
9171   \c_@@_available_keys_str
9172 }
9173 {
9174   The~available~keys~are~(in~alphabetic~order):~
9175   b,~
9176   baseline,~
9177   c,~
9178   cell~space~bottom~limit,~
9179   cell~space~limits,~
9180   cell~space~top~limit,~
9181   code~after,~
9182   code~for~first~col,~
9183   code~for~first~row,~
9184   code~for~last~col,~
9185   code~for~last~row,~

```

```

9186 colortbl-like,~
9187 columns-width,~
9188 corners,~
9189 create-extra-nodes,~
9190 create-medium-nodes,~
9191 create-large-nodes,~
9192 extra-left-margin,~
9193 extra-right-margin,~
9194 first-col,~
9195 first-row,~
9196 hlines,~
9197 hvlines,~
9198 hvlines-except-borders,~
9199 last-col,~
9200 last-row,~
9201 left-margin,~
9202 light-syntax,~
9203 name,~
9204 nullify-dots,~
9205 pgf-node-code,~
9206 renew-dots,~
9207 respect-arraystretch,~
9208 right-margin,~
9209 rules~(with~the~subkeys~'color'~and~'width'),~
9210 small,~
9211 t,~
9212 vlines,~
9213 xdots/color,~
9214 xdots/shorten-start,~
9215 xdots/shorten-end,~
9216 xdots/shorten~and~
9217 xdots/line-style.
9218 }

```

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).

```

9219 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9220 {
9221   Unknown~key.\\
9222   The~key~'\l_keys_key_str'~is~unknown~for~the~
9223   \@@_full_name_env:. \\
9224   That~key~will~be~ignored. \\
9225   \c_@@_available_keys_str
9226 }
9227 {
9228   The~available~keys~are~(in~alphabetic~order):~
9229   b,~
9230   baseline,~
9231   c,~
9232   cell-space-bottom-limit,~
9233   cell-space-limits,~
9234   cell-space-top-limit,~
9235   code-after,~
9236   code-for-first-col,~
9237   code-for-first-row,~
9238   code-for-last-col,~
9239   code-for-last-row,~
9240   colortbl-like,~
9241   columns-type,~
9242   columns-width,~
9243   corners,~
9244   create-extra-nodes,~
9245   create-medium-nodes,~
9246   create-large-nodes,~

```

```

9247     extra-left-margin,~
9248     extra-right-margin,~
9249     first-col,~
9250     first-row,~
9251     hlines,~
9252     hvlines,~
9253     hvlines-except-borders,~
9254     l,~
9255     last-col,~
9256     last-row,~
9257     left-margin,~
9258     light-syntax,~
9259     name,~
9260     nullify-dots,~
9261     pgf-node-code,~
9262     r,~
9263     renew-dots,~
9264     respect-arraystretch,~
9265     right-margin,~
9266     rules~(with~the~subkeys~'color'~and~'width'),~
9267     small,~
9268     t,~
9269     vlines,~
9270     xdots/color,~
9271     xdots/shorten-start,~
9272     xdots/shorten-end,~
9273     xdots/shorten-and~
9274     xdots/line-style.
9275 }
9276 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9277 {
9278     Unknown~key.\\
9279     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9280     \{NiceTabular\}. \\
9281     That~key~will~be~ignored. \\
9282     \c_@@_available_keys_str
9283 }
9284 {
9285     The~available~keys~are~(in~alphabetic~order):~
9286     b,~
9287     baseline,~
9288     c,~
9289     caption,~
9290     cell-space-bottom-limit,~
9291     cell-space-limits,~
9292     cell-space-top-limit,~
9293     code-after,~
9294     code-for-first-col,~
9295     code-for-first-row,~
9296     code-for-last-col,~
9297     code-for-last-row,~
9298     colortbl-like,~
9299     columns-width,~
9300     corners,~
9301     custom-line,~
9302     create-extra-nodes,~
9303     create-medium-nodes,~
9304     create-large-nodes,~
9305     extra-left-margin,~
9306     extra-right-margin,~
9307     first-col,~
9308     first-row,~
9309     hlines,~

```

```

9310     hvlines,~
9311     hvlines-except-borders,~
9312     label,~
9313     last-col,~
9314     last-row,~
9315     left-margin,~
9316     light-syntax,~
9317     name,~
9318     notes~(several~subkeys),~
9319     nullify-dots,~
9320     pgf-node-code,~
9321     renew-dots,~
9322     respect-arraystretch,~
9323     right-margin,~
9324     rounded-corners,~
9325     rules~(with~the~subkeys~'color'~and~'width'),~
9326     short-caption,~
9327     t,~
9328     tabularnote,~
9329     vlines,~
9330     xdots/color,~
9331     xdots/shorten-start,~
9332     xdots/shorten-end,~
9333     xdots/shorten-and~
9334     xdots/line-style.
9335 }

9336 \@@_msg_new:nnn { Duplicate-name }
9337 {
9338     Duplicate-name.\\
9339     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9340     the~same~environment~name~twice.~You~can~go~on,~but,~
9341     maybe,~you~will~have~incorrect~results~especially~
9342     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9343     message~again,~use~the~key~'allow-duplicate-names'~in~
9344     '\token_to_str:N \NiceMatrixOptions'.\\
9345     \c_@@_available_keys_str
9346 }
9347 {
9348     The~names~already~defined~in~this~document~are:~
9349     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9350 }

9351 \@@_msg_new:nn { Option-auto-for-columns-width }
9352 {
9353     Erroneous~use.\\
9354     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9355     That~key~will~be~ignored.
9356 }

```

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	115
20	The command <code>\RowStyle</code>	117
21	Colors of cells, rows and columns	119
22	The vertical and horizontal rules	128
23	The key corners	143
24	The environment <code>{NiceMatrixBlock}</code>	145
25	The extra nodes	146
26	The blocks	151
27	How to draw the dotted lines transparently	168
28	Automatic arrays	169
29	The redefinition of the command <code>\dotfill</code>	170
30	The command <code>\diagbox</code>	170

31	The keyword <code>\CodeAfter</code>	172
32	The delimiters in the preamble	173
33	The command <code>\SubMatrix</code>	174
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	182
35	The command <code>\ShowCellNames</code>	185
36	We process the options at package loading	188
37	About the package underscore	189
38	Error messages of the package	190