

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

July 17, 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 6.21a of `nicematrix`, at the date of 2023/07/17.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

2 Security test

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

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

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

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

```

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

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

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

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

```

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

```

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

```

3 Technical definitions

```

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

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

```

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

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

```

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

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes 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 `color-inside` 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_@@_tabular_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 } }
```

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

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

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

```
305 \bool_new:N \g_@@_delims_bool
306 \bool_gset_true:N \g_@@_delims_bool
```

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

The following boolean will be equal to true in the environments which have an environment (provided by the final user): {NiceTabular}, {NiceArray}, {pNiceArray}, etc.

```
307 \bool_new:N \l_@@_preamble_bool
308 \bool_set_true:N \l_@@_preamble_bool
```

We need a special treatment for {NiceMatrix} when vlines is not used, in order to retrieve \arraycolsep on both sides.

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

The following counter will count the environments {NiceMatrixBlock}.

```
310 \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.

```
311 \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).

```
312 \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.).

```
313 \dim_new:N \l_@@_col_width_dim
314 \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.

```
315 \int_new:N \g_@@_row_total_int
316 \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).

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

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

```
318 \int_new:N \l_@@_key_nb_rows_int
```

The following token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are `r`, `l`, `c` and `j`. For example, a column `p[1]{3cm}` will provide the value `l` for all the cells of the column.

```
319 \str_new:N \l_@@_hpos_cell_str
320 \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`.

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

Idem for the mono-row blocks.

```
322 \dim_new:N \g_@@_blocks_ht_dim
323 \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`.

```
324 \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}`).

```
325 \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`.

```
326 \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.

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

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

```
328 \bool_new:N \l_@@_notes_detect_duplicates_bool
329 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

If the user uses `{NiceTabular*}`, the width of the tabular (in the first argument of the environment `{NiceTabular*}`) will be stored in the following dimension.

```
330 \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).

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

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

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

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

```
333 \bool_new:N \g_@@_rotate_c_bool
```

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag.

```
334 \bool_new:N \l_@@_X_column_bool
335 \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 }`).

```
336 \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`.

```
337 \tl_new:N \l_@@_columns_type_tl
338 \hook_gput_code:nnn { begindocument } { . }
339 { \@@_set_preamble:Nn \l_@@_columns_type_tl { c } }
```

```
340 \cs_new_protected:Npn \@@_test_if_math_mode:
341 {
342   \if_mode_math: \else:
343     \@@_fatal:n { Outside-math-mode }
344   \fi:
345 }
```

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

```
346 \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.

```
347 \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”.

```
348 \colorlet { nicematrix-last-col } { . }
349 \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`).

```
350 \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*.

```
351 \tl_new:N \g_@@_com_or_env_str
352 \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).

```
353 \cs_new:Npn \@@_full_name_env:
354 {
355   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
356     { command \space \c_backslash_str \g_@@_name_env_str }
357     { environment \space \{ \g_@@_name_env_str \} }
358 }
```

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

```
359 \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.

```
360 \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.

```
361
```

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

```
362 \tl_new:N \g_@@_pre_code_before_tl
363 \tl_new:N \g_nicematrix_code_before_tl
```

The value of the key `code-before` will be added to the left of `\g_@@_pre_code_before_tl`. Idem for the code between `\CodeBefore` and `\Body`.

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

```
364 \tl_new:N \g_@@_pre_code_after_tl
365 \tl_new:N \g_nicematrix_code_after_tl
```

The `\CodeAfter` provided by the final user (with the key `code-after` or the keyword `\CodeAfter`) will be stored in the second token list.

```
366 \bool_new:N \l_@@_in_code_after_bool
```

The counters `\l_@@_old_iRow_int` and `\l_@@_old_jCol_int` will be used to save the values of the potential LaTeX counters `iRow` and `jCol`. These LaTeX counters will be restored at the end of the environment.

```
367 \int_new:N \l_@@_old_iRow_int
368 \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).

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

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

```
370 \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.

```
371 \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.

```
372 \bool_new:N \l_@@_X_columns_aux_bool
373 \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.

```
374 \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).

```
375 \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).

```
376 \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`).

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

```
378 \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).

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

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

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

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

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

```
383 \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.

```
384 \dim_zero_new:N \l_@@_tmpc_dim
```

```
385 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
386 \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”.

```
387 \dim_new:N \g_@@_width_last_col_dim
```

```
388 \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.

```
389 \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!).

```
390 \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: $\{imin\}\{jmin\}\{imax\}\{jmax\}\{name\}$.

```
391 \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.

```
392 \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.

```
393 \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`).

```
394 \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.

```
395 \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).

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

```
397 \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`).

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

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

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

```
401 \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.

```
402 \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).

```
403 \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`.

```
404 \tl_new:N \l_@@_fill_tl
405 \tl_new:N \l_@@_draw_tl
406 \seq_new:N \l_@@_tikz_seq
407 \clist_new:N \l_@@_borders_clist
408 \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.

```
409 \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`.

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

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

```
411 \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`).

```
412 \str_new:N \l_@@_hpos_block_str
413 \str_set:Nn \l_@@_hpos_block_str { c }
414 \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`.

```
415 \str_new:N \l_@@_vpos_of_block_str
416 \str_set:Nn \l_@@_vpos_of_block_str { c }
```

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

```
417 \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).

```
418 \bool_new:N \l_@@_vlines_block_bool
419 \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.

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

421 \dim_new:N \l_@@_submatrix_extra_height_dim
422 \dim_new:N \l_@@_submatrix_left_xshift_dim
423 \dim_new:N \l_@@_submatrix_right_xshift_dim
424 \clist_new:N \l_@@_hlines_clist
425 \clist_new:N \l_@@_vlines_clist
426 \clist_new:N \l_@@_submatrix_hlines_clist
427 \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}`).

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

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

```
429 \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`).

```
430 \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.

```
431 \int_new:N \l_@@_first_row_int
432 \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.

```
433 \int_new:N \l_@@_first_col_int
434 \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).

```
435 \int_new:N \l_@@_last_row_int
436 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
438 \bool_new:N \l_@@_last_col_without_value_bool
```

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.

- **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.

```
439 \int_new:N \l_@@_last_col_int
440 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
441 \bool_new:N \g_@@_last_col_found_bool
```

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

In the last column, we will raise the following flag (it will be used by `\OnlyMainNiceMatrix`).

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

Some utilities

```
443 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
444 {
445   \tl_set:Nn \l_tmpa_tl { #1 }
446   \tl_set:Nn \l_tmpb_tl { #2 }
447 }
```

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.

```
448 \cs_new_protected:Npn \@@_expand_clist:N #1
449 {
450   \clist_if_in:NnF #1 { all }
451   {
452     \clist_clear:N \l_tmpa_clist
453     \clist_map_inline:Nn #1
454     {
455       \tl_if_in:nnTF { ##1 } { - }
456       { \@@_cut_on_hyphen:w ##1 \q_stop }
457       {
458         \tl_set:Nn \l_tmpa_tl { ##1 }
459         \tl_set:Nn \l_tmpb_tl { ##1 }
460       }
461       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
462       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
463     }
464     \tl_set_eq:NN #1 \l_tmpa_clist
465   }
466 }
```

5 The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{\tabular}`.
- It's also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that's mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That's why:
 - The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.³
 - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each composant of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first composante is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker `\c_novalue_tl`).
 - During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the composantes of that sequence will be the same as for `\g_@@_notes_seq`.
 - After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

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

```

467 \newcounter { tabularnote }
468 \seq_new:N \g_@@_notes_seq
469 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

470 \tl_new:N \g_@@_tabularnote_tl

```

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

```

471 \seq_new:N \l_@@_notes_labels_seq
472 \newcounter{nicematrix_draft}
473 \cs_new_protected:Npn \@@_notes_format:n #1
474 {
475   \setcounter { nicematrix_draft } { #1 }
476   \@@_notes_style:n { nicematrix_draft }
477 }

```

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

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

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

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

```
479 \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`.

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

We define `\thetabularnote` because it will be used by LaTeX if the user want to reference a tabular which has been marked by a `\label`. The TeX group is for the case where the user has put an instruction such as `\color{red}` in `\@@_notes_style:n`.

```
481 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }
```

The tabular notes will be available for the final user only when `enumitem` is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by `enumitem` (a list `tabularnotes` in the general case and a list `tabularnotes*` if the key `para` is in force). However, we can test whether `enumitem` has been loaded only at the beginning of the document (we want to allow the user to load `enumitem` after `nicematrix`).

```
482 \hook_gput_code:nnn { begindocument } { . }
483 {
484   \IfPackageLoadedTF { enumitem }
485   {
```

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.

```
486     \newlist { tabularnotes } { enumerate } { 1 }
487     \setlist [ tabularnotes ]
488     {
489       topsep = 0pt ,
490       noitemsep ,
491       leftmargin = * ,
492       align = left ,
493       labelsep = 0pt ,
494       label =
495         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
496     }
497     \newlist { tabularnotes* } { enumerate* } { 1 }
498     \setlist [ tabularnotes* ]
499     {
500       afterlabel = \nobreak ,
501       itemjoin = \quad ,
502       label =
503         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
504     }
```

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`.

```
505     \NewDocumentCommand \tabularnote { o m }
506     {
507       \bool_if:nT { \cs_if_exist_p:N \@capttype || \l_@@_in_env_bool }
508       {
509         \bool_if:nTF { ! \l_@@_tabular_bool && \l_@@_in_env_bool }
510         { \@_error:n { tabularnote~forbidden } }
511         {
512           \bool_if:NTF \l_@@_in_caption_bool
513           { \@_tabularnote_caption:nn { #1 } { #2 } }
514           { \@_tabularnote:nn { #1 } { #2 } }
```

```

515         }
516     }
517 }
518 }
519 {
520     \NewDocumentCommand \tabularnote { o m }
521     {
522         \@@_error_or_warning:n { enumitem~not~loaded }
523         \@@_gredirect_none:n { enumitem~not~loaded }
524     }
525 }
526 }

```

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`.

```

527 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
528 {

```

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).

```

529     \int_zero:N \l_tmpa_int
530     \bool_if:NT \l_@@_notes_detect_duplicates_bool
531     {

```

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`.

```

532     \seq_map_indexed_inline:Nn \g_@@_notes_seq
533     {
534         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
535         {
536             \int_set:Nn \l_tmpa_int { ##1 }
537             \seq_map_break:
538         }
539     }
540     \int_compare:nNnF \l_tmpa_int = \c_zero_int
541     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
542 }
543 \int_compare:nNnT \l_tmpa_int = \c_zero_int
544 {
545     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
546     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
547 }
548 \seq_put_right:Nx \l_@@_notes_labels_seq
549 {
550     \tl_if_novalue:nTF { #1 }
551     {
552         \@@_notes_format:n
553         {
554             \int_eval:n
555             {
556                 \int_compare:nNnTF \l_tmpa_int = \c_zero_int
557                 \c@tabularnote
558                 \l_tmpa_int
559             }
560         }
561     }

```

```

562         { #1 }
563     }
564     \peek_meaning:NF \tabularnote
565     {

```

If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell when `\l_@@_hpos_cell_str` is equal to `c` or `r`.

```

566     \hbox_set:Nn \l_tmpa_box
567     {

```

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

```

568         \@@_notes_label_in_tabular:n
569         {
570             \seq_use:Nnnn
571             \l_@@_notes_labels_seq { , } { , } { , }
572         }
573     }

```

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

```

574     \int_gsub:Nn \c@tabularnote { 1 }
575     \int_set_eq:NN \l_tmpa_int \c@tabularnote
576     \refstepcounter { tabularnote }
577     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
578     { \int_gincr:N \c@tabularnote }
579     \seq_clear:N \l_@@_notes_labels_seq
580     \bool_lazy_or:nnTF
581     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { c } }
582     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { r } }
583     {
584         \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).

```

585         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
586     }
587     { \box_use:N \l_tmpa_box }
588 }
589 }

```

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`.

```

590 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
591 {
592     \bool_if:NTF \g_@@_caption_finished_bool
593     {
594         \int_compare:nNnT
595         \c@tabularnote = \g_@@_notes_caption_int
596         { \int_gzero:N \c@tabularnote }

```

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

```

597     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
598     { \@@_error:n { Identical~notes~in~caption } }
599 }
600 {

```

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

```
601     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
602     {
```

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

```
603         \bool_gset_true:N \g_@@_caption_finished_bool
604         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
605         \int_gzero:N \c@tabularnote
606     }
607     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
608 }
```

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!

```
609     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
610     \seq_put_right:Nx \l_@@_notes_labels_seq
611     {
612         \tl_if_novalue:nTF { #1 }
613         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
614         { #1 }
615     }
616     \peek_meaning:NF \tabularnote
617     {
618         \@@_notes_label_in_tabular:n
619         { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
620         \seq_clear:N \l_@@_notes_labels_seq
621     }
622 }

623 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
624 { \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.

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

```

645     }
646     { #1 }
647     { }
648   \end { pgfscope }
649 }

```

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.

```

650 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
651 {
652   \begin { pgfscope }
653   \pgfset
654   {
655     % outer~sep = \c_zero_dim ,
656     inner~sep = \c_zero_dim ,
657     minimum~size = \c_zero_dim
658   }
659   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
660   \pgfpointdiff { #3 } { #2 }
661   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
662   \pgfnode
663   { rectangle }
664   { center }
665   {
666     \vbox_to_ht:nn
667     { \dim_abs:n \l_tmpb_dim }
668     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
669   }
670   { #1 }
671   { }
672   \end { pgfscope }
673 }

```

7 The options

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

```

674 \tl_new:N \l_@@_caption_tl
675 \tl_new:N \l_@@_short_caption_tl
676 \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).

```

677 \bool_new:N \l_@@_caption_above_bool

```

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

```

678 \bool_new:N \l_@@_color_inside_bool

```

By default, the behaviour of `\cline` is changed in the environments of `nicematrix`: a `\cline` spreads the array by an amount equal to `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

```

679 \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`).

```
680 \dim_new:N \l_@@_cell_space_top_limit_dim
681 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
682 \bool_new:N \l_@@_xdots_h_labels_bool
```

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

```
683 \dim_new:N \l_@@_xdots_inter_dim
684 \hook_gput_code:nnn { begindocument } { . }
685 { \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).

```
686 \dim_new:N \l_@@_xdots_shorten_start_dim
687 \dim_new:N \l_@@_xdots_shorten_end_dim
688 \hook_gput_code:nnn { begindocument } { . }
689 {
690   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
691   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
692 }
```

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.

```
693 \dim_new:N \l_@@_xdots_radius_dim
694 \hook_gput_code:nnn { begindocument } { . }
695 { \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.

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

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

```
699 \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).

```
700 \tl_new:N \l_@@_baseline_tl
701 \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`).

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


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

```
703 \bool_new:N \l_@@_parallelize_diags_bool
704 \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.

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

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

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

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

```
710 \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).

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

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

```
712 \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.

```
713 \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”.

```
714 \bool_new:N \l_@@_medium_nodes_bool
715 \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).

```
716 \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`).

```
717 \dim_new:N \l_@@_left_margin_dim
718 \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`.

```
719 \dim_new:N \l_@@_extra_left_margin_dim
720 \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.

```
721 \tl_new:N \l_@@_end_of_row_tl
722 \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 “:”.

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

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

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

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

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

```
726 \keys_define:nn { NiceMatrix / xdots }
727 {
728   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
729   horizontal-labels .default:n = true ,
730   line-style .code:n =
731     {
732       \bool_lazy_or:nnTF
733         { \cs_if_exist_p:N \tikzpicture }
734         { \str_if_eq_p:nn { #1 } { standard } }
735         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
736         { \@_error:n { bad-option-for-line-style } }
737     } ,
738   line-style .value_required:n = true ,
739   color .tl_set:N = \l_@@_xdots_color_tl ,
740   color .value_required:n = true ,
741   shorten .code:n =
742     \hook_gput_code:nnn { begindocument } { . }
743     {
744       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
745       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
746     } ,
747   shorten-start .code:n =
748     \hook_gput_code:nnn { begindocument } { . }
749     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
750   shorten-end .code:n =
751     \hook_gput_code:nnn { begindocument } { . }
752     { \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.

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

```
764   down .tl_set:N = \l_@@_xdots_down_tl ,
765   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).

```
766   draw-first .code:n = \prg_do_nothing: ,
767   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
768 }
```

```
769 \keys_define:nn { NiceMatrix / rules }
770 {
771   color .tl_set:N = \l_@@_rules_color_tl ,
772   color .value_required:n = true ,
773   width .dim_set:N = \arrayrulewidth ,
774   width .value_required:n = true ,
775   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
776 }
```

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

```
777 \keys_define:nn { NiceMatrix / Global }
778 {
779   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
780   rounded-corners .default:n = 4 pt ,
781   custom-line .code:n = \@@_custom_line:n { #1 } ,
782   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
783   rules .value_required:n = true ,
784   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
785   standard-cline .default:n = true ,
786   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
787   cell-space-top-limit .value_required:n = true ,
788   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
789   cell-space-bottom-limit .value_required:n = true ,
790   cell-space-limits .meta:n =
791   {
792     cell-space-top-limit = #1 ,
793     cell-space-bottom-limit = #1 ,
794   } ,
795   cell-space-limits .value_required:n = true ,
796   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
797   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
798   light-syntax .default:n = true ,
799   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
800   end-of-row .value_required:n = true ,
801   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
802   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
803   last-row .int_set:N = \l_@@_last_row_int ,
804   last-row .default:n = -1 ,
805   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
806   code-for-first-col .value_required:n = true ,
807   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
808   code-for-last-col .value_required:n = true ,
809   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
810   code-for-first-row .value_required:n = true ,
811   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
812   code-for-last-row .value_required:n = true ,
813   hlines .clist_set:N = \l_@@_hlines_clist ,
814   vlines .clist_set:N = \l_@@_vlines_clist ,
815   hlines .default:n = all ,
```

```

816 vlines .default:n = all ,
817 vlines-in-sub-matrix .code:n =
818 {
819     \tl_if_single_token:nTF { #1 }
820     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
821     { \@@_error:n { One-letter-allowed } }
822 },
823 vlines-in-sub-matrix .value_required:n = true ,
824 hvlines .code:n =
825 {
826     \bool_set_true:N \l_@@_hvlines_bool
827     \clist_set:Nn \l_@@_vlines_clist { all }
828     \clist_set:Nn \l_@@_hlines_clist { all }
829 },
830 hvlines-except-borders .code:n =
831 {
832     \clist_set:Nn \l_@@_vlines_clist { all }
833     \clist_set:Nn \l_@@_hlines_clist { all }
834     \bool_set_true:N \l_@@_hvlines_bool
835     \bool_set_true:N \l_@@_except_borders_bool
836 },
837 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.

```

838 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
839 renew-dots .value_forbidden:n = true ,
840 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
841 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
842 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
843 create-extra-nodes .meta:n =
844 { create-medium-nodes , create-large-nodes } ,
845 left-margin .dim_set:N = \l_@@_left_margin_dim ,
846 left-margin .default:n = \arraycolsep ,
847 right-margin .dim_set:N = \l_@@_right_margin_dim ,
848 right-margin .default:n = \arraycolsep ,
849 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
850 margin .default:n = \arraycolsep ,
851 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
852 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
853 extra-margin .meta:n =
854 { extra-left-margin = #1 , extra-right-margin = #1 } ,
855 extra-margin .value_required:n = true ,
856 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
857 respect-arraystretch .default:n = true ,
858 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
859 pgf-node-code .value_required:n = true
860 }

```

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

```

861 \keys_define:nn { NiceMatrix / Env }
862 {
863     corners .clist_set:N = \l_@@_corners_clist ,
864     corners .default:n = { NW , SW , NE , SE } ,
865     code-before .code:n =
866     {
867         \tl_if_empty:nF { #1 }
868         {
869             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
870             \bool_set_true:N \l_@@_code_before_bool
871         }
872     }

```

```

872     } ,
873     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}`.

```

874     c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
875     t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
876     b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
877     baseline .tl_set:N = \l_@@_baseline_tl ,
878     baseline .value_required:n = true ,
879     columns-width .code:n =
880       \tl_if_eq:nnTF { #1 } { auto }
881       { \bool_set_true:N \l_@@_auto_columns_width_bool }
882       { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
883     columns-width .value_required:n = true ,
884     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.

```

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

```

```

929 enumitem-keys .value_required:n = true ,
930 enumitem-keys-para .code:n =
931 {
932     \hook_gput_code:nnn { begindocument } { . }
933     {
934         \IfPackageLoadedTF { enumitem }
935         { \setlist* [ tabularnotes* ] { #1 } }
936         { }
937     }
938 },
939 enumitem-keys-para .value_required:n = true ,
940 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
941 detect-duplicates .default:n = true ,
942 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
943 }
944 \keys_define:nn { NiceMatrix / delimiters }
945 {
946     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
947     max-width .default:n = true ,
948     color .tl_set:N = \l_@@_delimiters_color_tl ,
949     color .value_required:n = true ,
950 }

```

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

```

951 \keys_define:nn { NiceMatrix }
952 {
953     NiceMatrixOptions .inherit:n =
954     { NiceMatrix / Global } ,
955     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
956     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
957     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
958     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
959     SubMatrix / rules .inherit:n = NiceMatrix / rules ,
960     CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
961     CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
962     NiceMatrix .inherit:n =
963     {
964         NiceMatrix / Global ,
965         NiceMatrix / Env ,
966     } ,
967     NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
968     NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
969     NiceTabular .inherit:n =
970     {
971         NiceMatrix / Global ,
972         NiceMatrix / Env
973     } ,
974     NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
975     NiceTabular / rules .inherit:n = NiceMatrix / rules ,
976     NiceTabular / notes .inherit:n = NiceMatrix / notes ,
977     NiceArray .inherit:n =
978     {
979         NiceMatrix / Global ,
980         NiceMatrix / Env ,
981     } ,
982     NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
983     NiceArray / rules .inherit:n = NiceMatrix / rules ,
984     pNiceArray .inherit:n =
985     {
986         NiceMatrix / Global ,
987         NiceMatrix / Env ,

```

```

988     } ,
989     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
990     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
991 }

```

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

```

992 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
993 {
994     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
995     delimiters / color .value_required:n = true ,
996     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
997     delimiters / max-width .default:n = true ,
998     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
999     delimiters .value_required:n = true ,
1000     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
1001     width .value_required:n = true ,
1002     last-col .code:n =
1003         \tl_if_empty:nF { #1 }
1004         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1005         \int_zero:N \l_@@_last_col_int ,
1006     small .bool_set:N = \l_@@_small_bool ,
1007     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.

```

1008     renew-matrix .code:n = \@@_renew_matrix: ,
1009     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}`.

```

1010     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.

```

1011     columns-width .code:n =
1012         \tl_if_eq:nnTF { #1 } { auto }
1013         { \@@_error:n { Option-auto-for-columns-width } }
1014         { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1015     allow-duplicate-names .code:n =
1016         \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1017     allow-duplicate-names .value_forbidden:n = true ,
1018     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1019     notes .value_required:n = true ,
1020     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1021     sub-matrix .value_required:n = true ,
1022     matrix / columns-type .code:n =
1023         \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1024     matrix / columns-type .value_required:n = true ,
1025     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1026     caption-above .default:n = true ,
1027     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1028 }

```

\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.

```

1029 \NewDocumentCommand \NiceMatrixOptions { m }
1030 { \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.

```

1031 \keys_define:nn { NiceMatrix / NiceMatrix }
1032 {
1033   last-col .code:n = \tl_if_empty:nTF {#1}
1034   {
1035     \bool_set_true:N \l_@@_last_col_without_value_bool
1036     \int_set:Nn \l_@@_last_col_int { -1 }
1037   }
1038   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1039   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1040   columns-type .value_required:n = true ,
1041   l .meta:n = { columns-type = l } ,
1042   r .meta:n = { columns-type = r } ,
1043   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1044   delimiters / color .value_required:n = true ,
1045   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1046   delimiters / max-width .default:n = true ,
1047   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1048   delimiters .value_required:n = true ,
1049   small .bool_set:N = \l_@@_small_bool ,
1050   small .value_forbidden:n = true ,
1051   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1052 }

```

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

```

1053 \keys_define:nn { NiceMatrix / NiceArray }
1054 {

```

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.

```

1055   small .bool_set:N = \l_@@_small_bool ,
1056   small .value_forbidden:n = true ,
1057   last-col .code:n = \tl_if_empty:nF { #1 }
1058   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1059   \int_zero:N \l_@@_last_col_int ,
1060   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1061   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1062   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1063 }
1064 \keys_define:nn { NiceMatrix / pNiceArray }
1065 {
1066   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1067   last-col .code:n = \tl_if_empty:nF {#1}
1068   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1069   \int_zero:N \l_@@_last_col_int ,
1070   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1071   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1072   delimiters / color .value_required:n = true ,
1073   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1074   delimiters / max-width .default:n = true ,
1075   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1076   delimiters .value_required:n = true ,
1077   small .bool_set:N = \l_@@_small_bool ,
1078   small .value_forbidden:n = true ,
1079   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1080   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1081   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1082 }

```


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

```
1083 \keys_define:nn { NiceMatrix / NiceTabular }
1084 {
```

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.

```
1085   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1086           \bool_set_true:N \l_@@_width_used_bool ,
1087   width .value_required:n = true ,
1088   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1089   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1090   tabularnote .value_required:n = true ,
1091   caption .tl_set:N = \l_@@_caption_tl ,
1092   caption .value_required:n = true ,
1093   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1094   short-caption .value_required:n = true ,
1095   label .tl_set:N = \l_@@_label_tl ,
1096   label .value_required:n = true ,
1097   last-col .code:n = \tl_if_empty:NF {#1}
1098           { \@@_error:n { last-col~non-empty~for~NiceArray } }
1099           \int_zero:N \l_@@_last_col_int ,
1100   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1101   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1102   unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1103 }
```

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}).

```
1104 \cs_new_protected:Npn \@@_cell_begin:w
1105 {
```

\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.

```
1106   \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).

```
1107   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1108   \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.

```
1109   \int_compare:nNnT \c@jCol = 1
1110   { \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).

```
1111   \hbox_set:Nw \l_@@_cell_box
1112   \bool_if:NF \l_@@_tabular_bool
1113   {
1114     \c_math_toggle_token
1115     \bool_if:NT \l_@@_small_bool \scriptstyle
1116   }
```

```
1117 \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.

```
1118 \int_compare:nNnTF \c@iRow = 0
1119 {
1120   \int_compare:nNnT \c@jCol > 0
1121   {
1122     \l_@@_code_for_first_row_tl
1123     \xglobal \colorlet { nicematrix-first-row } { . }
1124   }
1125 }
1126 {
1127   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1128   {
1129     \l_@@_code_for_last_row_tl
1130     \xglobal \colorlet { nicematrix-last-row } { . }
1131   }
1132 }
1133 }
```

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.

```
1134 \cs_new_protected:Npn \@@_begin_of_row:
1135 {
1136   \int_gincr:N \c@iRow
1137   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1138   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1139   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1140   \pgfpicture
1141   \pgfrememberpicturepositiononpagetrue
1142   \pgfcoordinate
1143   { \@@_env: - row - \int_use:N \c@iRow - base }
1144   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1145   \str_if_empty:NF \l_@@_name_str
1146   {
1147     \pgfnodealias
1148     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1149     { \@@_env: - row - \int_use:N \c@iRow - base }
1150   }
1151   \endpgfpicture
1152 }
```

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.

```
1153 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1154 {
1155   \int_compare:nNnTF \c@iRow = 0
1156   {
1157     \dim_gset:Nn \g_@@_dp_row_zero_dim
1158     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1159     \dim_gset:Nn \g_@@_ht_row_zero_dim
1160     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1161   }
1162   {
1163     \int_compare:nNnT \c@iRow = 1
1164     {
```

```

1165         \dim_gset:Nn \g_@@_ht_row_one_dim
1166         { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1167     }
1168 }
1169 }
1170 \cs_new_protected:Npn \@@_rotate_cell_box:
1171 {
1172     \box_rotate:Nn \l_@@_cell_box { 90 }
1173     \bool_if:NTF \g_@@_rotate_c_bool
1174     {
1175         \hbox_set:Nn \l_@@_cell_box
1176         {
1177             \c_math_toggle_token
1178             \vcenter { \box_use:N \l_@@_cell_box }
1179             \c_math_toggle_token
1180         }
1181     }
1182     {
1183         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1184         {
1185             \vbox_set_top:Nn \l_@@_cell_box
1186             {
1187                 \vbox_to_zero:n { }
1188                 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1189                 \box_use:N \l_@@_cell_box
1190             }
1191         }
1192     }
1193     \bool_gset_false:N \g_@@_rotate_bool
1194     \bool_gset_false:N \g_@@_rotate_c_bool
1195 }
1196 \cs_new_protected:Npn \@@_adjust_size_box:
1197 {
1198     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1199     {
1200         \box_set_wd:Nn \l_@@_cell_box
1201         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1202         \dim_gzero:N \g_@@_blocks_wd_dim
1203     }
1204     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1205     {
1206         \box_set_dp:Nn \l_@@_cell_box
1207         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1208         \dim_gzero:N \g_@@_blocks_dp_dim
1209     }
1210     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1211     {
1212         \box_set_ht:Nn \l_@@_cell_box
1213         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1214         \dim_gzero:N \g_@@_blocks_ht_dim
1215     }
1216 }
1217 \cs_new_protected:Npn \@@_cell_end:
1218 {
1219     \@@_math_toggle_token:
1220     \hbox_set_end:
1221     \@@_cell_end_i:
1222 }
1223 \cs_new_protected:Npn \@@_cell_end_i:
1224 {

```

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.

```

1225 \g_@@_cell_after_hook_tl
1226 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1227 \@@_adjust_size_box:
1228 \box_set_ht:Nn \l_@@_cell_box
1229 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1230 \box_set_dp:Nn \l_@@_cell_box
1231 { \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”).

```

1232 \dim_gset:Nn \g_@@_max_cell_width_dim
1233 { \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”.

```

1234 \@@_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.

```

1235 \bool_if:NTF \g_@@_empty_cell_bool
1236 { \box_use_drop:N \l_@@_cell_box }
1237 {
1238   \bool_lazy_or:nnTF
1239     \g_@@_not_empty_cell_bool
1240     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1241     \@@_node_for_cell:
1242     { \box_use_drop:N \l_@@_cell_box }
1243 }
1244 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1245 \bool_gset_false:N \g_@@_empty_cell_bool
1246 \bool_gset_false:N \g_@@_not_empty_cell_bool
1247 }

```

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`).

```

1248 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1249 {
1250   \@@_math_toggle_token:
1251   \hbox_set_end:
1252   \bool_if:NF \g_@@_rotate_bool
1253   {
1254     \hbox_set:Nn \l_@@_cell_box
1255     {
1256       \makebox [ \l_@@_col_width_dim ] [ s ]

```

```

1257         { \hbox_unpack_drop:N \l_@@_cell_box }
1258     }
1259 }
1260 \@@_cell_end_i:
1261 }

```

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

```

1262 \pgfset
1263 {
1264     nicematrix / cell-node /.style =
1265     {
1266         inner~sep = \c_zero_dim ,
1267         minimum~width = \c_zero_dim
1268     }
1269 }
1270 \cs_new_protected:Npn \@@_node_for_cell:
1271 {
1272     \pgfpicture
1273     \pgfsetbaseline \c_zero_dim
1274     \pgfrememberpicturepositiononpagetrue
1275     \pgfset { nicematrix / cell-node }
1276     \pgfnode
1277     { rectangle }
1278     { base }
1279     {

```

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).

```

1280     \set@color
1281     \box_use_drop:N \l_@@_cell_box
1282 }
1283 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1284 { \l_@@_pgf_node_code_tl }
1285 \str_if_empty:NF \l_@@_name_str
1286 {
1287     \pgfnodealias
1288     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1289     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1290 }
1291 \endpgfpicture
1292 }

```

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.

```

1293 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1294 {
1295     \cs_new_protected:Npn \@@_patch_node_for_cell:
1296     {
1297         \hbox_set:Nn \l_@@_cell_box
1298         {
1299             \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1300             \hbox_overlap_left:n
1301             {
1302                 \pgfsys@markposition
1303                 { \@@_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.

```

1304         #1
1305     }
1306     \box_use:N \l_@@_cell_box

```

```

1307         \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1308         \hbox_overlap_left:n
1309         {
1310             \pgfsys@markposition
1311             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1312             #1
1313         }
1314     }
1315 }
1316 }

```

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

```

1317 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1318 {
1319     \@@_patch_node_for_cell:n
1320     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1321 }
1322 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

1323 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1324 {
1325     \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1326     { g_@@_#2 _ lines _ tl }
1327     {
1328         \use:c { @@ _ draw _ #2 : nnn }
1329         { \int_use:N \c@iRow }
1330         { \int_use:N \c@jCol }
1331         { \exp_not:n { #3 } }
1332     }
1333 }

1334 \cs_new_protected:Npn \@@_array:n
1335 {
1336     \bool_if:NTF \l_@@_tabular_bool
1337     { \dim_set_eq:NN \col@sep \tabcolsep }
1338     { \dim_set_eq:NN \col@sep \arraycolsep }
1339     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1340     { \cs_set_nopar:Npn \@halignto { } }
1341     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1342 \@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.

```

1343   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1344   }
1345   \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.

```

1346   \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1347   \cs_new_protected:Npn \@@_create_row_node:
1348   {
1349     \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1350     {
1351       \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1352       \@@_create_row_node_i:
1353     }
1354   }
1355   \cs_new_protected:Npn \@@_create_row_node_i:
1356   {

```

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

```

1357     \hbox
1358     {
1359       \bool_if:NT \l_@@_code_before_bool
1360       {
1361         \vtop
1362         {
1363           \skip_vertical:N 0.5\arrayrulewidth
1364           \pgfsys@markposition
1365           { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1366           \skip_vertical:N -0.5\arrayrulewidth
1367         }
1368       }
1369       \pgfpicture
1370       \pgfrememberpicturepositiononpagetrue
1371       \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1372       { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1373       \str_if_empty:NF \l_@@_name_str
1374       {
1375         \pgfnodealias
1376         { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1377         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1378       }
1379       \endpgfpicture
1380     }
1381   }

```

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

```

1382   \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1383   \cs_new_protected:Npn \@@_everycr_i:
1384   {
1385     \int_gzero:N \c@jCol
1386     \bool_gset_false:N \g_@@_after_col_zero_bool
1387     \bool_if:NF \g_@@_row_of_col_done_bool
1388     {
1389       \@@_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).

```

1390     \tl_if_empty:NF \l_@@_hlines_clist
1391     {
1392         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1393         {
1394             \exp_args:NNx
1395             \clist_if_in:NnT
1396             \l_@@_hlines_clist
1397             { \int_eval:n { \c@iRow + 1 } }
1398         }
1399     }

```

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.

```

1400     \int_compare:nNnT \c@iRow > { -1 }
1401     {
1402         \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@`.

```

1403         { \hrule height \arrayrulewidth width \c_zero_dim }
1404     }
1405 }
1406 }
1407 }
1408 }

```

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

```

1409 \cs_set_protected:Npn \@@_newcolumntype #1
1410 {
1411     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1412     \peek_meaning:NTF [
1413         { \newcol@ #1 }
1414         { \newcol@ #1 [ 0 ] }
1415     }

```

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

```

1416 \cs_set_protected:Npn \@@_renew_dots:
1417 {
1418     \cs_set_eq:NN \ldots \@@_Ldots
1419     \cs_set_eq:NN \cdots \@@_Cdots
1420     \cs_set_eq:NN \vdots \@@_Vdots
1421     \cs_set_eq:NN \ddots \@@_Ddots
1422     \cs_set_eq:NN \iddots \@@_Iddots
1423     \cs_set_eq:NN \dots \@@_Ldots
1424     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1425 }

```

When the key `color-inside` is used, the following code will be executed.

```

1426 \cs_new_protected:Npn \@@_colortbl_like:
1427 {
1428     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1429     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1430     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1431     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1432     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1433 }

```


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

```
1434 \cs_new_protected:Npn \@@_pre_array_ii:
1435 {
```

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

```
1436 \int_gzero:N \g_@@_total_X_weight_int
1437 \@@_expand_clist:N \l_@@_hlines_clist
1438 \@@_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 redefinition⁴.

```
1439 \IfPackageLoadedTF { booktabs }
1440 { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1441 { }
1442 \box_clear_new:N \l_@@_cell_box
1443 \normalbaselines
```

If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```
1444 \bool_if:NT \l_@@_small_bool
1445 {
1446     \cs_set_nopar:Npn \arraystretch { 0.47 }
1447     \dim_set:Nn \arraycolsep { 1.45 pt }
1448 }

1449 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1450 {
1451     \tl_put_right:Nn \@@_begin_of_row:
1452     {
1453         \pgfsys@markposition
1454         { \@@_env: - row - \int_use:N \c@iRow - base }
1455     }
1456 }
```

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`.

```
1457 \cs_set_nopar:Npn \ialign
1458 {
1459     \IfPackageLoadedTF { colortbl }
1460     {
1461         \CT@everycr
1462         {
1463             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1464             \@@_everycr:
1465         }
1466     }
1467     { \everycr { \@@_everycr: } }
1468     \tabskip = \c_zero_skip
```

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

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`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`.

```

1469      \dim_gzero_new:N \g_@@_dp_row_zero_dim
1470      \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1471      \dim_gzero_new:N \g_@@_ht_row_zero_dim
1472      \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1473      \dim_gzero_new:N \g_@@_ht_row_one_dim
1474      \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1475      \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1476      \dim_gzero_new:N \g_@@_ht_last_row_dim
1477      \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1478      \dim_gzero_new:N \g_@@_dp_last_row_dim
1479      \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`.

```

1480      \cs_set_eq:NN \ialign \@_old_ialign:
1481      \halign
1482    }

```

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).

```

1483      \cs_set_eq:NN \@_old_ldots \ldots
1484      \cs_set_eq:NN \@_old_cdots \cdots
1485      \cs_set_eq:NN \@_old_vdots \vdots
1486      \cs_set_eq:NN \@_old_ddots \ddots
1487      \cs_set_eq:NN \@_old_iddots \iddots
1488      \bool_if:NTF \l_@@_standard_cline_bool
1489        { \cs_set_eq:NN \cline \@_standard_cline }
1490        { \cs_set_eq:NN \cline \@_cline }
1491      \cs_set_eq:NN \Ldots \@_Ldots
1492      \cs_set_eq:NN \Cdots \@_Cdots
1493      \cs_set_eq:NN \Vdots \@_Vdots
1494      \cs_set_eq:NN \Ddots \@_Ddots
1495      \cs_set_eq:NN \Iddots \@_Iddots
1496      \cs_set_eq:NN \Hline \@_Hline:
1497      \cs_set_eq:NN \Hspace \@_Hspace:
1498      \cs_set_eq:NN \Hdotsfor \@_Hdotsfor:
1499      \cs_set_eq:NN \Vdotsfor \@_Vdotsfor:
1500      \cs_set_eq:NN \Block \@_Block:
1501      \cs_set_eq:NN \rotate \@_rotate:
1502      \cs_set_eq:NN \OnlyMainNiceMatrix \@_OnlyMainNiceMatrix:n
1503      \cs_set_eq:NN \dotfill \@_dotfill:
1504      \cs_set_eq:NN \CodeAfter \@_CodeAfter:
1505      \cs_set_eq:NN \diagbox \@_diagbox:nn
1506      \cs_set_eq:NN \NotEmpty \@_NotEmpty:
1507      \cs_set_eq:NN \RowStyle \@_RowStyle:n
1508      \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1509        { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1510      \bool_if:NT \l_@@_color_inside_bool \@_colortbl_like:
1511      \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.

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

```

1512 \cs_set_eq:NN \multicolumn \@_multicolumn:nnn
1513 \hook_gput_code:nnn { env / tabular / begin } { . }
1514 { \cs_set_eq:NN \multicolumn \@_old_multicolumn }

```

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

```

1515 \tl_if_exist:NT \l_@@_note_in_caption_tl
1516 {
1517   \tl_if_empty:NF \l_@@_note_in_caption_tl
1518   {
1519     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1520     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1521   }
1522 }

```

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).

```

1523 \seq_gclear:N \g_@@_multicolumn_cells_seq
1524 \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).

```

1525 \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`).

```

1526 \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.

```

1527 \int_gzero_new:N \g_@@_col_total_int
1528 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1529 \@@_renew_NC@rewrite@S:
1530 \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.

```

1531 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1532 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1533 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1534 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1535 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1536 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1537 \tl_gclear:N \g_nicematrix_code_before_tl
1538 \tl_gclear:N \g_@@_pre_code_before_tl
1539 }

```

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

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

```

1540 \cs_new_protected:Npn \@@_pre_array:
1541 {
1542   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1543   \int_gzero_new:N \c@iRow
1544   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1545   \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).

```

1546   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1547   {
1548     \bool_set_true:N \l_@@_last_row_without_value_bool
1549     \bool_if:NT \g_@@_aux_found_bool
1550     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1551   }
1552   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1553   {
1554     \bool_if:NT \g_@@_aux_found_bool
1555     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1556   }

```

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”.

```

1557   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1558   {
1559     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1560     {
1561       \dim_gset:Nn \g_@@_ht_last_row_dim
1562       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1563       \dim_gset:Nn \g_@@_dp_last_row_dim
1564       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1565     }
1566   }

1567   \seq_gclear:N \g_@@_cols_vlism_seq
1568   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1569   \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.

```

1570   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1571   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1572   \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.

```

1573   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1574   \@@_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.

```
1575 \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]`).

```
1576 \dim_zero_new:N \l_@@_left_delim_dim
1577 \dim_zero_new:N \l_@@_right_delim_dim
1578 \bool_if:NTF \g_@@_delims_bool
1579 {
```

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

```
1580 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1581 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1582 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1583 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1584 }
1585 {
1586 \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1587 \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1588 }
```

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).

```
1589 \hbox_set:Nw \l_@@_the_array_box
1590 \skip_horizontal:N \l_@@_left_margin_dim
1591 \skip_horizontal:N \l_@@_extra_left_margin_dim
1592 \c_math_toggle_token
1593 \bool_if:NTF \l_@@_light_syntax_bool
1594 { \use:c { @@-light-syntax } }
1595 { \use:c { @@-normal-syntax } }
1596 }
```

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

```
1597 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1598 {
1599 \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1600 \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...

```
1601 \@@_pre_array:
1602 }
```

9 The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed (that command will be used only once and is present only for legibility).

```
1603 \cs_new_protected:Npn \@@_pre_code_before:
1604 {
```

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).

```

1605 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1606 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1607 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1608 \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.

```

1609 \pgfsys@markposition { \@@_env: - position }
1610 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1611 \pgfpicture
1612 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1613 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1614 {
1615   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1616   \pgfcoordinate { \@@_env: - row - ##1 }
1617   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1618 }

```

Now, the recreation of the `col` nodes.

```

1619 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1620 {
1621   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1622   \pgfcoordinate { \@@_env: - col - ##1 }
1623   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1624 }

```

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

```

1625 \@@_create_diag_nodes:

```

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

```

1626 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1627 \endpgfpicture

```

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

```

1628 \@@_create_blocks_nodes:
1629 \IfPackageLoadedTF { tikz }
1630 {
1631   \tikzset
1632   {
1633     every~picture / .style =
1634     { overlay , name~prefix = \@@_env: - }
1635   }
1636 }
1637 { }
1638 \cs_set_eq:NN \cellcolor \@@_cellcolor
1639 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1640 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1641 \cs_set_eq:NN \rowcolor \@@_rowcolor
1642 \cs_set_eq:NN \rowcolors \@@_rowcolors
1643 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1644 \cs_set_eq:NN \arraycolor \@@_arraycolor
1645 \cs_set_eq:NN \columncolor \@@_columncolor
1646 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1647 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1648 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1649 }

```

```

1650 \cs_new_protected:Npn \@@_exec_code_before:
1651 {
1652   \seq_gclear_new:N \g_@@_colors_seq
1653   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1654   \group_begin:

```

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

```

1655   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token

```

The following code is a security for the case the user has used `babel` with the option `spanish`: in that case, the characters `<` (de code ASCII 60) and `>` are activated and Tikz is not able to solve the problem (even with the Tikz library `babel`).

```

1656   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1657   {
1658     \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1659     \@@_rescan_for_spanish:N \l_@@_code_before_tl
1660   }

```

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`.

```

1661   \exp_last_unbraced:NV \@@_CodeBefore_keys:
1662   \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.

```

1663   \@@_actually_color:
1664   \l_@@_code_before_tl
1665   \q_stop
1666   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1667   \group_end:
1668   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1669     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1670 }

```

```

1671 \keys_define:nn { NiceMatrix / CodeBefore }
1672 {
1673   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1674   create-cell-nodes .default:n = true ,
1675   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1676   sub-matrix .value_required:n = true ,
1677   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1678   delimiters / color .value_required:n = true ,
1679   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1680 }
1681 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1682 {
1683   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1684   \@@_CodeBefore:w
1685 }

```

We have extracted the options of the keyword `\CodeBefore` in order to see whether the key `create-cell-nodes` has been used. Now, you can execute the rest of the `\CodeBefore`, excepted, of course, if we are in the first compilation.

```

1686 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1687 {
1688   \bool_if:NT \g_@@_aux_found_bool
1689   {

```

```

1690     \@@_pre_code_before:
1691     #1
1692   }
1693 }

```

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.

```

1694 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1695 {
1696   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1697   {
1698     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1699     \pgfcoordinate { \@@_env: - row - ##1 - base }
1700     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1701     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1702     {
1703       \cs_if_exist:cT
1704       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1705       {
1706         \pgfsys@getposition
1707         { \@@_env: - ##1 - ####1 - NW }
1708         \@@_node_position:
1709         \pgfsys@getposition
1710         { \@@_env: - ##1 - ####1 - SE }
1711         \@@_node_position_i:
1712         \@@_pgf_rect_node:nnn
1713         { \@@_env: - ##1 - ####1 }
1714         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1715         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1716       }
1717     }
1718   }
1719   \int_step_inline:nn \c@iRow
1720   {
1721     \pgfnodealias
1722     { \@@_env: - ##1 - last }
1723     { \@@_env: - ##1 - \int_use:N \c@jCol }
1724   }
1725   \int_step_inline:nn \c@jCol
1726   {
1727     \pgfnodealias
1728     { \@@_env: - last - ##1 }
1729     { \@@_env: - \int_use:N \c@iRow - ##1 }
1730   }
1731   \@@_create_extra_nodes:
1732 }

1733 \cs_new_protected:Npn \@@_create_blocks_nodes:
1734 {
1735   \pgfpicture
1736   \pgf@relevantforpicturesizefalse
1737   \pgfrememberpicturepositiononpagetrue
1738   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1739   { \@@_create_one_block_node:nnnnn ##1 }
1740   \endpgfpicture
1741 }

```

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

```

1742 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1743 {
1744   \tl_if_empty:nF { #5 }
1745   {
1746     \@@_qpoint:n { col - #2 }
1747     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1748     \@@_qpoint:n { #1 }
1749     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1750     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1751     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1752     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1753     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1754     \@@_pgf_rect_node:nnnnn
1755     { \@@_env: - #5 }
1756     { \dim_use:N \l_tmpa_dim }
1757     { \dim_use:N \l_tmpb_dim }
1758     { \dim_use:N \l_@@_tmpc_dim }
1759     { \dim_use:N \l_@@_tmpd_dim }
1760   }
1761 }

1762 \cs_new_protected:Npn \@@_patch_for_revtext:
1763 {
1764   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1765   \cs_set_eq:NN \insert@column \insert@column@array
1766   \cs_set_eq:NN \@classx \@classx@array
1767   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1768   \cs_set_eq:NN \@arraycr \@arraycr@array
1769   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1770   \cs_set_eq:NN \array \array@array
1771   \cs_set_eq:NN \@array \@array@array
1772   \cs_set_eq:NN \@tabular \@tabular@array
1773   \cs_set_eq:NN \@mkpream \@mkpream@array
1774   \cs_set_eq:NN \endarray \endarray@array
1775   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1776   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1777 }

```

10 The environment {NiceArrayWithDelims}

```

1778 \NewDocumentEnvironment { NiceArrayWithDelims }
1779 { m m O { } m ! O { } t \CodeBefore }
1780 {
1781   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1782   \@@_provide_pgfsyspdfmark:
1783   \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.

```

1784   \bgroup

1785   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1786   \tl_gset:Nn \g_@@_right_delim_tl { #2 }

```

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).

```

1787 \tl_gset:Nn \g_@@_preamble_tl { #4 }

1788 \int_gzero:N \g_@@_block_box_int
1789 \dim_zero:N \g_@@_width_last_col_dim
1790 \dim_zero:N \g_@@_width_first_col_dim
1791 \bool_gset_false:N \g_@@_row_of_col_done_bool
1792 \str_if_empty:NT \g_@@_name_env_str
1793 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1794 \bool_if:NTF \l_@@_tabular_bool
1795 \mode_leave_vertical:
1796 \@@_test_if_math_mode:
1797 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1798 \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 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.

```

1799 \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.

```

1800 \cs_if_exist:NT \tikz@library@external@loaded
1801 {
1802   \tikzexternaldisable
1803   \cs_if_exist:NT \ifstandalone
1804   { \tikzset { external / optimize = false } }
1805 }

```

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

```

1806 \int_gincr:N \g_@@_env_int
1807 \bool_if:NF \l_@@_block_auto_columns_width_bool
1808 { \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`).

```

1809 \seq_gclear:N \g_@@_blocks_seq
1810 \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`.

```

1811 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1812 \seq_gclear:N \g_@@_pos_of_xdots_seq
1813 \tl_gclear_new:N \g_@@_code_before_tl
1814 \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.

```

1815 \bool_gset_false:N \g_@@_aux_found_bool
1816 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1817 {
1818   \bool_gset_true:N \g_@@_aux_found_bool
1819   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1820 }

```

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.

```

1821 \tl_gclear:N \g_@@_aux_tl
1822 \tl_if_empty:NF \g_@@_code_before_tl

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1823 {
1824   \bool_set_true:N \l_@@_code_before_bool
1825   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1826 }
1827 \tl_if_empty:NF \g_@@_pre_code_before_tl
1828 { \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.

```

1829   \bool_if:NTF \g_@@_delims_bool
1830   { \keys_set:nn { NiceMatrix / pNiceArray } }
1831   { \keys_set:nn { NiceMatrix / NiceArray } }
1832   { #3 , #5 }

1833   \@@_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:.

```

1834   \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1835 }

```

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

```

1836 {
1837   \bool_if:NTF \l_@@_light_syntax_bool
1838   { \use:c { end @@-light-syntax } }
1839   { \use:c { end @@-normal-syntax } }
1840   \c_math_toggle_token
1841   \skip_horizontal:N \l_@@_right_margin_dim
1842   \skip_horizontal:N \l_@@_extra_right_margin_dim
1843   \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.

```

1844   \bool_if:NT \l_@@_width_used_bool
1845   {
1846     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1847     { \@@_error_or_warning:n { width~without~X~columns } }
1848   }

```

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 .

```

1849   \int_compare:nNnT \g_@@_total_X_weight_int > 0
1850   {
1851     \tl_gput_right:Nx \g_@@_aux_tl
1852     {
1853       \bool_set_true:N \l_@@_X_columns_aux_bool
1854       \dim_set:Nn \l_@@_X_columns_dim
1855       {
1856         \dim_compare:nNnTF
1857         {
1858           \dim_abs:n
1859           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1860         }
1861         <
1862         { 0.001 pt }
1863         { \dim_use:N \l_@@_X_columns_dim }

```

```

1864         {
1865             \dim_eval:n
1866             {
1867                 ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1868                 / \int_use:N \g_@@_total_X_weight_int
1869                 + \l_@@_X_columns_dim
1870             }
1871         }
1872     }
1873 }
1874 }

```

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

```

1875 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1876 {
1877     \bool_if:NF \l_@@_last_row_without_value_bool
1878     {
1879         \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1880         {
1881             \@@_error:n { Wrong~last~row }
1882             \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1883         }
1884     }
1885 }

```

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

```

1886 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1887 \bool_if:nTF \g_@@_last_col_found_bool
1888 { \int_gdecr:N \c@jCol }
1889 {
1890     \int_compare:nNnT \l_@@_last_col_int > { -1 }
1891     { \@@_error:n { last~col~not~used } }
1892 }

```

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

```

1893 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1894 \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. 85).

```

1895 \int_compare:nNnT \l_@@_first_col_int = 0
1896 {
1897     \skip_horizontal:N \col@sep
1898     \skip_horizontal:N \g_@@_width_first_col_dim
1899 }

```

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

```

1900 \bool_if:nTF { ! \g_@@_delims_bool }
1901 {
1902     \str_case:VnF \l_@@_baseline_tl
1903     {
1904         b \@@_use_arraybox_with_notes_b:
1905         c \@@_use_arraybox_with_notes_c:
1906     }
1907     \@@_use_arraybox_with_notes:
1908 }

```

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

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

```

1909 {
1910   \int_compare:nNnTF \l_@@_first_row_int = 0
1911   {
1912     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1913     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1914   }
1915   { \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

```

1916   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1917   {
1918     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1919     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1920   }
1921   { \dim_zero:N \l_tmpb_dim }
1922   \hbox_set:Nn \l_tmpa_box
1923   {
1924     \c_math_toggle_token
1925     \@@_color:V \l_@@_delimiters_color_tl
1926     \exp_after:wN \left \g_@@_left_delim_tl
1927     \vcenter
1928     {

```

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.

```

1929       \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1930       \hbox
1931       {
1932         \bool_if:NTF \l_@@_tabular_bool
1933         { \skip_horizontal:N -\tabcolsep }
1934         { \skip_horizontal:N -\arraycolsep }
1935         \@@_use_arraybox_with_notes_c:
1936         \bool_if:NTF \l_@@_tabular_bool
1937         { \skip_horizontal:N -\tabcolsep }
1938         { \skip_horizontal:N -\arraycolsep }
1939       }

```

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

```

1940       \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1941     }

```

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.

```

1942     \@@_color:V \l_@@_delimiters_color_tl
1943     \exp_after:wN \right \g_@@_right_delim_tl
1944     \c_math_toggle_token
1945   }

```

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.

```

1946   \bool_if:NTF \l_@@_delimiters_max_width_bool
1947   {
1948     \@@_put_box_in_flow_bis:nn
1949     \g_@@_left_delim_tl \g_@@_right_delim_tl
1950   }
1951   \@@_put_box_in_flow:
1952 }

```

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).

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. 86).

```

1953 \bool_if:NT \g_@@_last_col_found_bool
1954 {
1955     \skip_horizontal:N \g_@@_width_last_col_dim
1956     \skip_horizontal:N \col@sep
1957 }
1958 \bool_if:NT \l_@@_preamble_bool
1959 {
1960     \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1961     { \@@_warning_gredirect_none:n { columns-not-used } }
1962 }
1963 \@@_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.

```

1964 \egroup

```

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

```

1965 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1966 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1967 \iow_now:Nx \@mainaux
1968 {
1969     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1970     { \exp_not:V \g_@@_aux_tl }
1971 }
1972 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1973 \bool_if:NT \c_@@_footnote_bool \endsavenotes
1974 }

```

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.

```

1975 \cs_new_protected:Npn \@@_transform_preamble:
1976 {
1977     \bool_if:NT \l_@@_preamble_bool \@@_transform_preamble_i:
1978     \@@_transform_preamble_ii:
1979 }

1980 \cs_new_protected:Npn \@@_transform_preamble_i:
1981 {

```

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*

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

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.

```

1982 \group_begin:
1983 \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1984 \@@_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).

```

1985 \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).

```

1986 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1987 \@tempswatrue

```

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

```

1988 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
1989 \int_gzero:N \c@jCol
1990 \tl_gclear:N \g_@@_preamble_tl

```

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

```

1991 \bool_gset_false:N \g_tmpb_bool
1992 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1993 {
1994   \tl_gset:Nn \g_@@_preamble_tl
1995   { ! { \skip_horizontal:N \arrayrulewidth } }
1996 }
1997 {
1998   \clist_if_in:NnT \l_@@_vlines_clist 1
1999   {
2000     \tl_gset:Nn \g_@@_preamble_tl
2001     { ! { \skip_horizontal:N \arrayrulewidth } }
2002   }
2003 }

```

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 `vlism`).

```

2004 \seq_clear:N \g_@@_cols_vlism_seq

```

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

```

2005 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2006 \int_zero:N \l_tmpa_int

```

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

```

2007 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
2008 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

Remark that `\g_@@_static_num_of_col_int` will stay equal to zero in the environments without preamble since we are in a code that is executed only in the environments *with* preamble.

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

```

2009 \bool_if:NT \l_@@_color_inside_bool
2010 {
2011   \regex_replace_all:NnN
2012   \c_@@_columncolor_regex
2013   { \c { @@_columncolor_preamble } }
2014   \g_@@_preamble_tl
2015 }

```

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

```
2016   \group_end:
2017 }
```

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`.

```
2018 \cs_new_protected:Npn \@@_transform_preamble_ii:
2019 {
2020 %
2021 % \medskip
2022 % If there was delimiters at the beginning or at the end of the preamble, the
2023 % environment \{NiceArray\} is transformed into an environment \{xNiceMatrix\}.
2024 % \begin{macrocode}
2025 \bool_lazy_or:nnT
2026 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
2027 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2028 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2029 \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).

```
2030 \int_compare:nNnTF \l_@@_first_col_int = 0
2031 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
2032 {
2033   \bool_lazy_all:nT
2034   {
2035     { \bool_not_p:n \g_@@_delims_bool }
2036     { \bool_not_p:n \l_@@_tabular_bool }
2037     { \tl_if_empty_p:N \l_@@_vlines_clist }
2038     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2039   }
2040   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
2041 }
2042 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2043 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
2044 {
2045   \bool_lazy_all:nT
2046   {
2047     { \bool_not_p:n \g_@@_delims_bool }
2048     { \bool_not_p:n \l_@@_tabular_bool }
2049     { \tl_if_empty_p:N \l_@@_vlines_clist }
2050     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2051   }
2052   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
2053 }
```

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`).

```
2054 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2055 {
2056   \tl_gput_right:Nn \g_@@_preamble_tl
2057   { > { \@@_error_too_much_cols: } 1 }
2058 }
2059 }
```


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

```

2060 \cs_new_protected:Npn \@@_patch_preamble:n #1
2061 {
2062   \str_case:nnF { #1 }
2063   {
2064     c      { \@@_patch_preamble_i:n #1 }
2065     l      { \@@_patch_preamble_i:n #1 }
2066     r      { \@@_patch_preamble_i:n #1 }
2067     >      { \@@_patch_preamble_xiv:n }
2068     !      { \@@_patch_preamble_ii:nn #1 }
2069     @      { \@@_patch_preamble_ii:nn #1 }
2070     |      { \@@_patch_preamble_iii:n #1 }
2071     p      { \@@_patch_preamble_iv:n #1 }
2072     b      { \@@_patch_preamble_iv:n #1 }
2073     m      { \@@_patch_preamble_iv:n #1 }
2074     \@@_V: { \@@_patch_preamble_v:n }
2075     V      { \@@_patch_preamble_v:n }
2076     \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
2077     \@@_W: { \@@_patch_preamble_vi:nnnn { \@@_special_W: } #1 }
2078     \@@_S: { \@@_patch_preamble_vii:n }
2079     (      { \@@_patch_preamble_viii:nn #1 }
2080     [      { \@@_patch_preamble_viii:nn #1 }
2081     \{      { \@@_patch_preamble_viii:nn #1 }
2082     \left   { \@@_patch_preamble_viii:nn }
2083     )      { \@@_patch_preamble_ix:nn #1 }
2084     ]      { \@@_patch_preamble_ix:nn #1 }
2085     \}      { \@@_patch_preamble_ix:nn #1 }
2086     \right  { \@@_patch_preamble_ix:nn }
2087     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}`.

```

2088   \@@_X { \@@_patch_preamble_x:n }
2089   \q_stop { }
2090 }
2091 {
2092   \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2093   {
2094     \seq_gput_right:Nx \g_@@_cols_vlism_seq
2095     { \int_eval:n { \c@jCol + 1 } }
2096     \tl_gput_right:Nx \g_@@_preamble_tl
2097     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2098     \@@_patch_preamble:n
2099   }

```

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).

```

2100   {
2101     \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2102     {
2103       \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2104       \@@_patch_preamble:n
2105     }
2106     {
2107       \tl_if_eq:nnT { #1 } { S }
2108       { \@@_fatal:n { unknown~column~type~S } }
2109       { \@@_fatal:nn { unknown~column~type } { #1 } }

```

```

2110     }
2111   }
2112 }
2113 }

```

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

```

2114 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2115 {
2116   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2117   \tl_gclear:N \g_@@_pre_cell_tl
2118   \tl_gput_right:Nn \g_@@_preamble_tl
2119   {
2120     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2121     #1
2122     < \@@_cell_end:
2123   }

```

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

```

2124   \int_gincr:N \c@jCol
2125   \@@_patch_preamble_xi:n
2126 }

```

For >, ! and @

```

2127 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2128 {
2129   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2130   \@@_patch_preamble:n
2131 }

```

For |

```

2132 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2133 {
2134   \int_incr:N \l_tmpa_int
2135   \@@_patch_preamble_iii_i:n
2136 }

```

\l_tmpa_int is the number of successive occurrences of |

```

2137 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2138 {
2139   \str_if_eq:nnTF { #1 } |
2140   { \@@_patch_preamble_iii:n | }
2141   {
2142     \dim_set:Nn \l_tmpa_dim
2143     {
2144       \arrayrulewidth * \l_tmpa_int
2145       + \doublerulesep * ( \l_tmpa_int - 1 )
2146     }
2147     \tl_gput_right:Nx \g_@@_preamble_tl
2148     {

```

Here, the command \dim_eval:n is mandatory.

```

2149       \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_tmpa_dim } } }
2150     }
2151     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2152     {
2153       \@@_vline:n
2154       {
2155         position = \int_eval:n { \c@jCol + 1 } ,
2156         multiplicity = \int_use:N \l_tmpa_int ,
2157         total-width = \dim_use:N \l_tmpa_dim
2158       }

```

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.

```

2159     }
2160     \int_zero:N \l_tmpa_int
2161     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2162     \@@_patch_preamble:n #1
2163   }
2164 }

2165 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2166 {
2167   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2168   \@@_patch_preamble:n
2169 }

2170 \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.

```

2171 \keys_define:nn { WithArrows / p-column }
2172 {
2173   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2174   r .value_forbidden:n = true ,
2175   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2176   c .value_forbidden:n = true ,
2177   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2178   l .value_forbidden:n = true ,
2179   R .code:n =
2180     \IfPackageLoadedTF { ragged2e }
2181     { \str_set:Nn \l_@@_hpos_col_str { R } }
2182     {
2183       \@@_error_or_warning:n { ragged2e~not-loaded }
2184       \str_set:Nn \l_@@_hpos_col_str { r }
2185     } ,
2186   R .value_forbidden:n = true ,
2187   L .code:n =
2188     \IfPackageLoadedTF { ragged2e }
2189     { \str_set:Nn \l_@@_hpos_col_str { L } }
2190     {
2191       \@@_error_or_warning:n { ragged2e~not-loaded }
2192       \str_set:Nn \l_@@_hpos_col_str { l }
2193     } ,
2194   L .value_forbidden:n = true ,
2195   C .code:n =
2196     \IfPackageLoadedTF { ragged2e }
2197     { \str_set:Nn \l_@@_hpos_col_str { C } }
2198     {
2199       \@@_error_or_warning:n { ragged2e~not-loaded }
2200       \str_set:Nn \l_@@_hpos_col_str { c }
2201     } ,
2202   C .value_forbidden:n = true ,
2203   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2204   S .value_forbidden:n = true ,
2205   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2206   p .value_forbidden:n = true ,
2207   t .meta:n = p ,
2208   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2209   m .value_forbidden:n = true ,
2210   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2211   b .value_forbidden:n = true ,
2212 }

```

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

```

2213 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2214 {
2215   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2216   \@@_patch_preamble_iv_i:n
2217 }

2218 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2219 {
2220   \str_if_eq:nnTF { #1 } { [ ]
2221     { \@@_patch_preamble_iv_ii:w [ ]
2222       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2223     }
2224   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2225     { \@@_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.

```

2226 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2227 {

```

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).

```

2228   \str_set:Nn \l_@@_hpos_col_str { j }
2229   \tl_set:Nn \l_tmpa_tl { #1 }
2230   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2231   \@@_keys_p_column:V \l_tmpa_tl
2232   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2233 }

2234 \cs_new_protected:Npn \@@_keys_p_column:n #1
2235 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2236 \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*.

```

2237 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2238 {
2239   \use:x
2240   {
2241     \@@_patch_preamble_iv_v:nnnnnnnn
2242     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2243     { \dim_eval:n { #1 } }
2244     {

```

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.

```

2245       \str_if_eq:VnTF \l_@@_hpos_col_str j
2246       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2247       {
2248         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2249         { \str_lowercase:V \l_@@_hpos_col_str }
2250       }
2251   \str_case:Vn \l_@@_hpos_col_str
2252   {
2253     c { \exp_not:N \centering }
2254     l { \exp_not:N \raggedright }
2255     r { \exp_not:N \raggedleft }
2256     C { \exp_not:N \Centering }
2257     L { \exp_not:N \RaggedRight }
2258     R { \exp_not:N \RaggedLeft }

```

```

2259     }
2260   }
2261   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2262   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2263   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2264   { #2 }
2265   {
2266     \str_case:VnF \l_@@_hpos_col_str
2267     {
2268       { j } { c }
2269       { si } { c }
2270     }

```

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

```

2271     { \str_lowercase:V \l_@@_hpos_col_str }
2272   }
2273 }

```

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

```

2274   \int_gincr:N \c@jCol
2275   \@@_patch_preamble_xi:n
2276 }

```

#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*.

```

2277 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2278 {
2279   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2280   { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2281   { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2282   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2283   \tl_gclear:N \g_@@_pre_cell_tl
2284   \tl_gput_right:Nn \g_@@_preamble_tl
2285   {
2286     > {

```

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.

```

2287   \dim_set:Nn \l_@@_col_width_dim { #2 }
2288   \@@_cell_begin:w
2289   \begin { #7 } [ #1 ] { #2 }

```

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

```

2290   \everypar
2291   {
2292     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2293     \everypar { }
2294   }

```

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

```

2295   #3

```

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

```

2296         \g_@@_row_style_tl
2297         \arraybackslash
2298         #5
2299     }
2300     #8
2301     < {
2302         #6

```

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

```

2303         \@finalstrut \@arstrutbox
2304         % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
2305         \end { #7 }

```

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

```

2306         #4
2307         \@@_cell_end:
2308     }
2309 }
2310 }

```

```

2311 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2312 {
2313     \peek_meaning:NT \unskip
2314     {
2315         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2316         {
2317             \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.

```

2318         \skip_horizontal:N \l_@@_col_width_dim
2319     }
2320 }
2321 #1
2322 }
2323 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2324 {
2325     \peek_meaning:NT \_siunitx_table_skip:n
2326     {
2327         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2328         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2329     }
2330     #1
2331 }

```

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

```

2332 \cs_new_protected:Npn \@@_center_cell_box:
2333 {

```

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

```

2334     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2335     {
2336         \int_compare:nNnT
2337         { \box_ht:N \l_@@_cell_box }
2338         >

```

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).

```

2339     { \box_ht:N \strutbox }
2340     {
2341       \hbox_set:Nn \l_@@_cell_box
2342       {
2343         \box_move_down:nn
2344         {
2345           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2346             + \baselineskip ) / 2
2347         }
2348         { \box_use:N \l_@@_cell_box }
2349       }
2350     }
2351   }
2352 }

```

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

```

2353 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2354 {
2355   \str_if_eq:nnTF { #1 } { [ ] }
2356   { \@@_patch_preamble_v_i:w [ ] }
2357   { \@@_patch_preamble_v_i:w [ ] { #1 } }
2358 }
2359 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2360 { \@@_patch_preamble_v_ii:nn { #1 } }
2361 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2362 {
2363   \str_set:Nn \l_@@_vpos_col_str { p }
2364   \str_set:Nn \l_@@_hpos_col_str { j }
2365   \tl_set:Nn \l_tmpa_tl { #1 }
2366   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2367   \@@_keys_p_column:V \l_tmpa_tl
2368   \IfPackageLoadedTF { varwidth }
2369   { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2370   {
2371     \@@_error_or_warning:n { varwidth~not~loaded }
2372     \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2373   }
2374 }

```

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.

```

2375 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2376 {
2377   \str_if_eq:nnTF { #3 } { s }
2378   { \@@_patch_preamble_vi_i:nnnn { #1 } { #4 } }
2379   { \@@_patch_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2380 }

```

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.

```

2381 \cs_new_protected:Npn \@@_patch_preamble_vi_i:nnnn #1 #2
2382 {
2383   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2384   \tl_gclear:N \g_@@_pre_cell_tl

```

```

2385 \tl_gput_right:Nn \g_@@_preamble_tl
2386 {
2387   > {
2388     \dim_set:Nn \l_@@_col_width_dim { #2 }
2389     \@@_cell_begin:w
2390     \str_set:Nn \l_@@_hpos_cell_str { c }
2391   }
2392   c
2393   < {
2394     \@@_cell_end_for_ws:
2395     #1
2396     \@@_adjust_size_box:
2397     \box_use_drop:N \l_@@_cell_box
2398   }
2399 }
2400 \int_gincr:N \c@jCol
2401 \@@_patch_preamble_xi:n
2402 }

```

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

```

2403 \cs_new_protected:Npn \@@_patch_preamble_viii:nnnn #1 #2 #3 #4
2404 {
2405   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2406   \tl_gclear:N \g_@@_pre_cell_tl
2407   \tl_gput_right:Nn \g_@@_preamble_tl
2408   {
2409     > {

```

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.

```

2410       \dim_set:Nn \l_@@_col_width_dim { #4 }
2411       \hbox_set:Nw \l_@@_cell_box
2412       \@@_cell_begin:w
2413       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2414     }
2415     c
2416     < {
2417       \@@_cell_end:
2418       \hbox_set_end:
2419       #1
2420       \@@_adjust_size_box:
2421       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2422     }
2423   }

```

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

```

2424   \int_gincr:N \c@jCol
2425   \@@_patch_preamble_xi:n
2426 }

```

```

2427 \cs_new_protected:Npn \@@_special_W:
2428 {
2429   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2430   { \@@_warning:n { W~warning } }
2431 }

```

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).

```

2432 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2433 {
2434   \str_if_eq:nnTF { #1 } { [ ] }
2435   { \@@_patch_preamble_vii_i:w [ ] }
2436   { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2437 }

```



```

2438 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2439 { \@@_patch_preamble_vii_ii:n { #1 } }
2440 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2441 {
2442   \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2443   {
2444     \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2445     \tl_gclear:N \g_@@_pre_cell_tl
2446     \tl_gput_right:Nn \g_@@_preamble_tl
2447     {
2448       > {
2449         \@@_cell_begin:w
2450         \keys_set:nn { siunitx } { #1 }
2451         \siunitx_cell_begin:w
2452       }
2453       c
2454       < { \siunitx_cell_end: \@@_cell_end: }
2455     }

```

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

```

2456   \int_gincr:N \c@jCol
2457   \@@_patch_preamble_xi:n
2458 }
2459 { \@@_fatal:n { Version~of~siunitx~too~old } }
2460 }

```

For (, [and \{.

```

2461 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2462 {
2463   \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.

```

2464   \int_compare:nNnTF \c@jCol = \c_zero_int
2465   {
2466     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2467     {

```

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

```

2468     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2469     \tl_gset:Nn \g_@@_right_delim_tl { . }
2470     \@@_patch_preamble:n #2
2471   }
2472   {
2473     \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2474     \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2475   }
2476 }
2477 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2478 }
2479 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2480 {
2481   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2482   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }
2483   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2484   {
2485     \@@_error:nn { delimiter~after~opening } { #2 }
2486     \@@_patch_preamble:n
2487   }
2488   { \@@_patch_preamble:n #2 }
2489 }

```

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

```

2490 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2491 {
2492   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter-with-small } }
2493   \tl_if_in:nnTF { ) ] \} } { #2 }
2494   { \@@_patch_preamble_ix_i:nnn #1 #2 }
2495   {
2496     \tl_if_eq:nnTF { \q_stop } { #2 }
2497     {
2498       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2499       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2500       {
2501         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2502         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2503         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2504         \@@_patch_preamble:n #2
2505       }
2506     }
2507     {
2508       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2509       { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2510       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2511       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2512       \@@_patch_preamble:n #2
2513     }
2514   }
2515 }

2516 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2517 {
2518   \tl_if_eq:nnTF { \q_stop } { #3 }
2519   {
2520     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2521     {
2522       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2523       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2524       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2525       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2526     }
2527     {
2528       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2529       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2530       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2531       \@@_error:nn { double-closing-delimiter } { #2 }
2532     }
2533   }
2534   {
2535     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2536     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2537     \@@_error:nn { double-closing-delimiter } { #2 }
2538     \@@_patch_preamble:n #3
2539   }
2540 }

```

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.

```

2541 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2542 {
2543   \str_if_eq:nnTF { #1 } { [ ]
2544     { \@@_patch_preamble_x_i:w [ ]

```

```

2545     { \@@_patch_preamble_x_i:w [ ] #1 }
2546   }

2547 \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2548   { \@@_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).

```

2549 \keys_define:nn { WithArrows / X-column }
2550   { 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.

```

2551 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2552   {

```

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).

```

2553   \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).

```

2554   \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.

```

2555   \int_zero_new:N \l_@@_weight_int
2556   \int_set:Nn \l_@@_weight_int { 1 }
2557   \tl_set:Nn \l_tmpa_tl { #1 }
2558   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2559   \@@_keys_p_column:V \l_tmpa_tl
2560   \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2561   \int_compare:nNnT \l_@@_weight_int < 0
2562   {
2563     \@@_error_or_warning:n { negative-weight }
2564     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2565   }
2566   \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).

```

2567   \bool_if:NTF \l_@@_X_columns_aux_bool
2568   {
2569     \exp_args:Nnx
2570     \@@_patch_preamble_iv_iv:nn
2571     { \l_@@_weight_int \l_@@_X_columns_dim }
2572     { minipage }
2573   }
2574   {
2575     \tl_gput_right:Nn \g_@@_preamble_tl
2576     {
2577       > {
2578         \@@_cell_begin:w
2579         \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.

```

2580   \NotEmpty

```

The following code will nullify the box of the cell.

```
2581         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2582         { \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`.

```
2583         \begin { minipage } { 5 cm } \arraybackslash
2584     }
2585     c
2586     < {
2587         \end { minipage }
2588         \@@_cell_end:
2589     }
2590 }
2591 \int_gincr:N \c@jCol
2592 \@@_patch_preamble_xi:n
2593 }
2594 }
```

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 `@{...}`.

```
2595 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2596 {
2597     \str_if_eq:nnTF { #1 } { < }
2598     \@@_patch_preamble_xiii:n
2599     {
2600         \str_if_eq:nnTF { #1 } { @ }
2601         \@@_patch_preamble_xv:n
2602         {
2603             \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2604             {
2605                 \tl_gput_right:Nn \g_@@_preamble_tl
2606                 { ! { \skip_horizontal:N \arrayrulewidth } }
2607             }
2608             {
2609                 \exp_args:NNx
2610                 \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2611                 {
2612                     \tl_gput_right:Nn \g_@@_preamble_tl
2613                     { ! { \skip_horizontal:N \arrayrulewidth } }
2614                 }
2615             }
2616             \@@_patch_preamble:n { #1 }
2617         }
2618     }
2619 }
2620 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2621 {
2622     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2623     \@@_patch_preamble_xi:n
2624 }
```

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.

```
2625 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2626 {
2627     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2628     {
2629         \tl_gput_right:Nn \g_@@_preamble_tl
2630         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2631     }
2632 }
```

```

2633 \exp_args:NNx
2634 \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2635 {
2636     \tl_gput_right:Nn \g_@@_preamble_tl
2637     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2638 }
2639 { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2640 }
2641 \@@_patch_preamble:n
2642 }

2643 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2644 {
2645     \group_begin:
2646     \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2647     \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2648     \@temptokena { #2 }
2649     \@tempswatrue
2650     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2651     \tl_gclear:N \g_@@_preamble_tl
2652     \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
2653     \group_end:
2654     \tl_set_eq:NN #1 \g_@@_preamble_tl
2655 }

```

12 The redefinition of `\multicolumn`

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

```

2656 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2657 {

```

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`.

```

2658     \multispan { #1 }
2659     \begingroup
2660     \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
2661     \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2662     \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

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

```

2663     \@temptokena = { #2 }
2664     \@tempswatrue
2665     \@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`.

```

2666     \tl_gclear:N \g_@@_preamble_tl
2667     \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

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

```

2668     \exp_args:NV \mkpream \g_@@_preamble_tl
2669     \@addtopreamble \empty
2670     \endgroup

```

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

```

2671 \int_compare:nNnT { #1 } > 1
2672 {
2673   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2674   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2675   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2676   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2677   {
2678     {
2679       \int_compare:nNnTF \c@jCol = 0
2680       { \int_eval:n { \c@iRow + 1 } }
2681       { \int_use:N \c@iRow }
2682     }
2683     { \int_eval:n { \c@jCol + 1 } }
2684     {
2685       \int_compare:nNnTF \c@jCol = 0
2686       { \int_eval:n { \c@iRow + 1 } }
2687       { \int_use:N \c@iRow }
2688     }
2689     { \int_eval:n { \c@jCol + #1 } }
2690     { } % for the name of the block
2691   }
2692 }

```

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

```

2693 \cs_set:Npn \@sharp { #3 }
2694 \@arstrut
2695 \@preamble
2696 \null

```

We add some lines.

```

2697 \int_gadd:Nn \c@jCol { #1 - 1 }
2698 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2699 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2700 \ignorespaces
2701 }

```

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`.

```

2702 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2703 {
2704   \str_case:nnF { #1 }
2705   {
2706     c { \@@_patch_m_preamble_i:n #1 }
2707     l { \@@_patch_m_preamble_i:n #1 }
2708     r { \@@_patch_m_preamble_i:n #1 }
2709     > { \@@_patch_m_preamble_ii:nn #1 }
2710     ! { \@@_patch_m_preamble_ii:nn #1 }
2711     @ { \@@_patch_m_preamble_ii:nn #1 }
2712     | { \@@_patch_m_preamble_iii:n #1 }
2713     p { \@@_patch_m_preamble_iv:nnn t #1 }
2714     m { \@@_patch_m_preamble_iv:nnn c #1 }
2715     b { \@@_patch_m_preamble_iv:nnn b #1 }
2716     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2717     \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2718     \q_stop { }
2719   }
2720   {
2721     \tl_if_eq:nnT { #1 } { S }
2722     { \@@_fatal:n { unknown-column-type-S } }
2723     { \@@_fatal:nn { unknown-column-type } { #1 } }

```

```

2724     }
2725 }

```

For c, l and r

```

2726 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2727 {
2728   \tl_gput_right:Nn \g_@@_preamble_tl
2729   {
2730     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2731     #1
2732     < \@@_cell_end:
2733   }

```

We test for the presence of a <.

```

2734   \@@_patch_m_preamble_x:n
2735 }

```

For >, ! and @

```

2736 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2737 {
2738   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2739   \@@_patch_m_preamble:n
2740 }

```

For |

```

2741 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2742 {
2743   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2744   \@@_patch_m_preamble:n
2745 }

```

For p, m and b

```

2746 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2747 {
2748   \tl_gput_right:Nn \g_@@_preamble_tl
2749   {
2750     > {
2751       \@@_cell_begin:w
2752       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2753       \mode_leave_vertical:
2754       \arraybackslash
2755       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2756     }
2757     c
2758     < {
2759       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2760       \end { minipage }
2761       \@@_cell_end:
2762     }
2763   }

```

We test for the presence of a <.

```

2764   \@@_patch_m_preamble_x:n
2765 }

```

For w and W

```

2766 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2767 {
2768   \tl_gput_right:Nn \g_@@_preamble_tl
2769   {
2770     > {
2771       \dim_set:Nn \l_@@_col_width_dim { #4 }
2772       \hbox_set:Nw \l_@@_cell_box
2773       \@@_cell_begin:w

```

```

2774         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2775     }
2776     c
2777     < {
2778         \@@_cell_end:
2779         \hbox_set_end:
2780         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2781         #1
2782         \@@_adjust_size_box:
2783         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2784     }
2785 }

```

We test for the presence of a <.

```

2786     \@@_patch_m_preamble_x:n
2787 }

```

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

```

2788 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2789 {
2790     \str_if_eq:nnTF { #1 } { < }
2791     \@@_patch_m_preamble_ix:n
2792     { \@@_patch_m_preamble:n { #1 } }
2793 }
2794 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2795 {
2796     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2797     \@@_patch_m_preamble_x:n
2798 }

```

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).

```

2799 \cs_new_protected:Npn \@@_put_box_in_flow:
2800 {
2801     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2802     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2803     \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2804     { \box_use_drop:N \l_tmpa_box }
2805     \@@_put_box_in_flow_i:
2806 }

```

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).

```

2807 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2808 {
2809     \pgfpicture
2810     \@@_qpoint:n { row - 1 }
2811     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2812     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2813     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2814     \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).

```

2815     \str_if_in:NnTF \l_@@_baseline_tl { line- }
2816     {
2817         \int_set:Nn \l_tmpa_int
2818         {
2819             \str_range:Nnn
2820             \l_@@_baseline_tl
2821             6

```



```

2822         { \tl_count:V \l_@@_baseline_tl }
2823     }
2824     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2825 }
2826 {
2827     \str_case:VnF \l_@@_baseline_tl
2828     {
2829         { t } { \int_set:Nn \l_tmpa_int 1 }
2830         { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2831     }
2832     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2833     \bool_lazy_or:nnT
2834     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2835     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2836     {
2837         \@@_error:n { bad-value-for-baseline }
2838         \int_set:Nn \l_tmpa_int 1
2839     }
2840     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2841     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2842 }
2843 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2844     \endpgfpicture
2845     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2846     \box_use_drop:N \l_tmpa_box
2847 }

```

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).

```

2848 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2849 {

```

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.

```

2850     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2851     {
2852         \box_set_wd:Nn \l_@@_the_array_box
2853         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2854     }

```

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

```

2855     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2856     \bool_if:NT \l_@@_caption_above_bool
2857     {
2858         \tl_if_empty:NF \l_@@_caption_tl
2859         {
2860             \bool_set_false:N \g_@@_caption_finished_bool
2861             \int_gzero:N \c@tabularnote
2862             \@@_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).

```

2863         \int_compare:nNnT \g_@@_notes_caption_int > 0
2864         {
2865             \tl_gput_right:Nx \g_@@_aux_tl
2866             {

```

```

2867         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2868         { \int_use:N \g_@@_notes_caption_int }
2869     }
2870     \int_gzero:N \g_@@_notes_caption_int
2871 }
2872 }
2873 }

```

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

```

2874 \hbox
2875 {
2876     \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.

```

2877     \@@_create_extra_nodes:
2878     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2879 }

```

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

```

2880 \bool_lazy_any:nT
2881 {
2882     { ! \seq_if_empty_p:N \g_@@_notes_seq }
2883     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2884     { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2885 }
2886 \@@_insert_tabularnotes:
2887 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2888 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2889 \end { minipage }
2890 }

```

```

2891 \cs_new_protected:Npn \@@_insert_caption:
2892 {
2893     \tl_if_empty:NF \l_@@_caption_tl
2894     {
2895         \cs_if_exist:NTF \@capytype
2896         { \@@_insert_caption_i: }
2897         { \@@_error:n { caption~outside~float } }
2898     }
2899 }

```

```

2900 \cs_new_protected:Npn \@@_insert_caption_i:
2901 {
2902     \group_begin:

```

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

```

2903     \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.

```

2904 \IfPackageLoadedTF { floatrow }
2905 { \cs_set_eq:NN \@makecaption \FR@makecaption }
2906 { }
2907 \tl_if_empty:NTF \l_@@_short_caption_tl
2908 { \caption }

```

```

2909     { \caption [ \l_@@_short_caption_tl ] }
2910     { \l_@@_caption_tl }

```

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That's why, when the same tabular note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of tabular notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

```

2911     \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
2912     {
2913         \bool_gset_true:N \g_@@_caption_finished_bool
2914         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabulernote
2915         \int_gzero:N \c@tabulernote
2916     }
2917     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2918     \group_end:
2919 }

```

```

2920 \cs_new_protected:Npn \@@_tabulernote_error:n #1
2921 {
2922     \@@_error_or_warning:n { tabulernote~below~the~tabular }
2923     \@@_gredirect_none:n { tabulernote~below~the~tabular }
2924 }
2925 \cs_new_protected:Npn \@@_insert_tabularnotes:
2926 {
2927     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2928     \int_set:Nn \c@tabulernote { \seq_count:N \g_@@_notes_seq }
2929     \skip_vertical:N 0.65ex

```

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

```

2930     \group_begin:
2931     \l_@@_notes_code_before_tl
2932     \tl_if_empty:NF \g_@@_tabulernote_tl
2933     {
2934         \g_@@_tabulernote_tl \par
2935         \tl_gclear:N \g_@@_tabulernote_tl
2936     }

```

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.

```

2937     \int_compare:nNnT \c@tabulernote > 0
2938     {
2939         \bool_if:NTF \l_@@_notes_para_bool
2940         {
2941             \begin { tabularnotes* }
2942                 \seq_map_inline:Nn \g_@@_notes_seq
2943                     { \@@_one_tabulernote:nn ##1 }
2944                 \strut
2945             \end { tabularnotes* }

```

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

```

2946         \par
2947     }
2948     {
2949         \tabularnotes
2950             \seq_map_inline:Nn \g_@@_notes_seq
2951                 { \@@_one_tabulernote:nn ##1 }
2952             \strut
2953         \endtabularnotes
2954     }
2955 }
2956 \unskip
2957 \group_end:
2958 \bool_if:NT \l_@@_notes_bottomrule_bool

```

```

2959 {
2960   \IfPackageLoadedTF { booktabs }
2961   {

```

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

```

2962     \skip_vertical:N \aboverulesep

```

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

```

2963     { \CT@arc@ \hrule height \heavyrulewidth }
2964   }
2965   { \@@_error_or_warning:n { bottomrule-without-booktabs } }
2966 }
2967 \l_@@_notes_code_after_tl
2968 \seq_gclear:N \g_@@_notes_seq
2969 \seq_gclear:N \g_@@_notes_in_caption_seq
2970 \int_gzero:N \c@tabularnote
2971 }

```

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

```

2972 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
2973 {
2974   \tl_if_novalue:nTF { #1 }
2975   { \item }
2976   { \item [ \@@_notes_label_in_list:n { #1 } ] }
2977 }

```

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`.

```

2978 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2979 {
2980   \pgfpicture
2981     \@@_qpoint:n { row - 1 }
2982     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2983     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2984     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2985   \endpgfpicture
2986   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2987   \int_compare:nNnT \l_@@_first_row_int = 0
2988   {
2989     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2990     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2991   }
2992   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2993 }

```

Now, the general case.

```

2994 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2995 {

```

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

```

2996   \tl_if_eq:NnT \l_@@_baseline_tl { t }
2997   { \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`.

```

2998   \pgfpicture
2999     \@@_qpoint:n { row - 1 }
3000     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3001     \str_if_in:NnTF \l_@@_baseline_tl { line- }
3002     {

```

```

3003 \int_set:Nn \l_tmpa_int
3004 {
3005   \str_range:Nnn
3006     \l_@@_baseline_tl
3007     6
3008     { \tl_count:V \l_@@_baseline_tl }
3009 }
3010 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3011 }
3012 {
3013   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3014   \bool_lazy_or:nnT
3015     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3016     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3017     {
3018       \@@_error:n { bad~value~for~baseline }
3019       \int_set:Nn \l_tmpa_int 1
3020     }
3021   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3022 }
3023 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3024 \endpgfpicture
3025 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3026 \int_compare:nNnT \l_@@_first_row_int = 0
3027 {
3028   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3029   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3030 }
3031 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3032 }

```

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.

```

3033 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3034 {

```

We will compute the real width of both delimiters used.

```

3035   \dim_zero_new:N \l_@@_real_left_delim_dim
3036   \dim_zero_new:N \l_@@_real_right_delim_dim
3037   \hbox_set:Nn \l_tmpb_box
3038     {
3039     \c_math_toggle_token
3040     \left #1
3041     \vcenter
3042     {
3043       \vbox_to_ht:nn
3044         { \box_ht_plus_dp:N \l_tmpa_box }
3045         { }
3046     }
3047     \right .
3048     \c_math_toggle_token
3049   }
3050   \dim_set:Nn \l_@@_real_left_delim_dim
3051     { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
3052   \hbox_set:Nn \l_tmpb_box
3053     {
3054     \c_math_toggle_token
3055     \left .
3056     \vbox_to_ht:nn
3057       { \box_ht_plus_dp:N \l_tmpa_box }
3058       { }
3059     \right #2

```

```

3060     \c_math_toggle_token
3061   }
3062   \dim_set:Nn \l_@@_real_right_delim_dim
3063   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3064   \skip_horizontal:N \l_@@_left_delim_dim
3065   \skip_horizontal:N -\l_@@_real_left_delim_dim
3066   \@@_put_box_in_flow:
3067   \skip_horizontal:N \l_@@_right_delim_dim
3068   \skip_horizontal:N -\l_@@_real_right_delim_dim
3069 }

```

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).

```

3070 \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).

```

3071 {
3072   \peek_remove_spaces:n
3073   {
3074     \peek_meaning:NTF \end
3075     \@@_analyze_end:Nn
3076     {
3077       \@@_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`).

```

3078       \@@_array:V \g_@@_preamble_tl
3079     }
3080   }
3081 }
3082 {
3083   \@@_create_col_nodes:
3084   \endarray
3085 }

```

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

```

3086 \NewDocumentEnvironment { @@-light-syntax } { b }
3087 {

```

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.

```

3088   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
3089   \tl_map_inline:nn { #1 }
3090   {
3091     \str_if_eq:nnT { ##1 } { & }
3092     { \@@_fatal:n { ampersand~in~light-syntax } }
3093     \str_if_eq:nnT { ##1 } { \ }
3094     { \@@_fatal:n { double-backslash~in~light-syntax } }
3095   }

```

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`.

```
3096 \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3097 }
```

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.

```
3098 {
3099 \@@_create_col_nodes:
3100 \endarray
3101 }
3102 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3103 {
3104 \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).

```
3105 \seq_clear_new:N \l_@@_rows_seq
```

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

```
3106 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3107 \seq_set_split:NVn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3108 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3109 \tl_if_empty:NF \l_tmpa_tl
3110 { \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.

```
3111 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3112 { \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`).

```
3113 \tl_clear_new:N \l_@@_new_body_tl
3114 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3115 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3116 \@@_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).

```
3117 \seq_map_inline:Nn \l_@@_rows_seq
3118 {
3119 \tl_put_right:Nn \l_@@_new_body_tl { \\ }
3120 \@@_line_with_light_syntax:n { ##1 }
3121 }
3122 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3123 {
3124 \int_set:Nn \l_@@_last_col_int
3125 { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3126 }
```

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.

```
3127 \@@_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`).

```
3128 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3129 }
3130 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3131 {
3132   \seq_clear_new:N \l_@@_cells_seq
3133   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3134   \int_set:Nn \l_@@_nb_cols_int
3135   {
3136     \int_max:nn
3137     \l_@@_nb_cols_int
3138     { \seq_count:N \l_@@_cells_seq }
3139   }
3140   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3141   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3142   \seq_map_inline:Nn \l_@@_cells_seq
3143   { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3144 }
3145 \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`.

```
3146 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3147 {
3148   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3149   { \@@_fatal:n { empty~environment } }
3150 }
```

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

```
3150 \end { #2 }
3151 }
```

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).

```
3152 \cs_new:Npn \@@_create_col_nodes:
3153 {
3154   \crrcr
3155   \int_compare:nNnT \l_@@_first_col_int = 0
3156   {
3157     \omit
3158     \hbox_overlap_left:n
3159     {
3160       \bool_if:NT \l_@@_code_before_bool
3161       { \pgfsys@markposition { \@@_env: - col - 0 } }
3162       \pgfpicture
3163       \pgfrememberpicturepositiononpagetrue
3164       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointright
3165       \str_if_empty:NF \l_@@_name_str
3166       { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3167       \endpgfpicture
3168       \skip_horizontal:N 2\col@sep
3169       \skip_horizontal:N \g_@@_width_first_col_dim
3170     }
3171     &
3172   }
3173   \omit
3174 }
```


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

```
3174 \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`).

```
3175 \int_compare:nNnTF \l_@@_first_col_int = 0
3176 {
3177   \bool_if:NT \l_@@_code_before_bool
3178   {
3179     \hbox
3180     {
3181       \skip_horizontal:N -0.5\arrayrulewidth
3182       \pgfsys@markposition { \@@_env: - col - 1 }
3183       \skip_horizontal:N 0.5\arrayrulewidth
3184     }
3185   }
3186   \pgfpicture
3187   \pgfrememberpicturepositiononpagetrue
3188   \pgfcoordinate { \@@_env: - col - 1 }
3189   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3190   \str_if_empty:NF \l_@@_name_str
3191   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3192   \endpgfpicture
3193 }
3194 {
3195   \bool_if:NT \l_@@_code_before_bool
3196   {
3197     \hbox
3198     {
3199       \skip_horizontal:N 0.5\arrayrulewidth
3200       \pgfsys@markposition { \@@_env: - col - 1 }
3201       \skip_horizontal:N -0.5\arrayrulewidth
3202     }
3203   }
3204   \pgfpicture
3205   \pgfrememberpicturepositiononpagetrue
3206   \pgfcoordinate { \@@_env: - col - 1 }
3207   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3208   \str_if_empty:NF \l_@@_name_str
3209   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3210   \endpgfpicture
3211 }
```

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.

```
3212 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3213 \bool_if:NF \l_@@_auto_columns_width_bool
3214 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3215 {
3216   \bool_lazy_and:nnTF
3217   \l_@@_auto_columns_width_bool
3218   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3219   { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3220   { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3221   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3222 }
3223 \skip_horizontal:N \g_tmpa_skip
3224 \hbox
3225 {
```

```

3226 \bool_if:NT \l_@@_code_before_bool
3227 {
3228     \hbox
3229     {
3230         \skip_horizontal:N -0.5\arrayrulewidth
3231         \pgfsys@markposition { \@@_env: - col - 2 }
3232         \skip_horizontal:N 0.5\arrayrulewidth
3233     }
3234 }
3235 \pgfpicture
3236 \pgfrememberpicturepositiononpagetrue
3237 \pgfcoordinate { \@@_env: - col - 2 }
3238 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3239 \str_if_empty:NF \l_@@_name_str
3240 { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3241 \endpgfpicture
3242 }

```

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.

```

3243 \int_gset:Nn \g_tmpa_int 1
3244 \bool_if:NTF \g_@@_last_col_found_bool
3245 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3246 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3247 {
3248     &
3249     \omit
3250     \int_gincr:N \g_tmpa_int

```

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

```

3251 \skip_horizontal:N \g_tmpa_skip
3252 \bool_if:NT \l_@@_code_before_bool
3253 {
3254     \hbox
3255     {
3256         \skip_horizontal:N -0.5\arrayrulewidth
3257         \pgfsys@markposition
3258         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3259         \skip_horizontal:N 0.5\arrayrulewidth
3260     }
3261 }

```

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

```

3262 \pgfpicture
3263 \pgfrememberpicturepositiononpagetrue
3264 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3265 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3266 \str_if_empty:NF \l_@@_name_str
3267 {
3268     \pgfnodealias
3269     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3270     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3271 }
3272 \endpgfpicture
3273 }

3274 &
3275 \omit

```

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

```

3276 \int_compare:nNnT \g_@@_col_total_int = 1
3277 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3278 \skip_horizontal:N \g_tmpa_skip

```

```

3279 \int_gincr:N \g_tmpa_int
3280 \bool_lazy_all:nT
3281 {
3282   { \bool_not_p:n \g_@@_delims_bool }
3283   { \bool_not_p:n \l_@@_tabular_bool }
3284   { \clist_if_empty_p:N \l_@@_vlines_clist }
3285   { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3286   { ! \l_@@_bar_at_end_of_pream_bool }
3287 }
3288 { \skip_horizontal:N -\col@sep }
3289 \bool_if:NT \l_@@_code_before_bool
3290 {
3291   \hbox
3292   {
3293     \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.

```

3294     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3295     { \skip_horizontal:N -\arraycolsep }
3296     \pgfsys@markposition
3297     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3298     \skip_horizontal:N 0.5\arrayrulewidth
3299     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3300     { \skip_horizontal:N \arraycolsep }
3301   }
3302 }
3303 \pgfpicture
3304 \pgfrememberpicturepositiononpagetrue
3305 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3306 {
3307   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3308   {
3309     \pgfpoint
3310     { - 0.5 \arrayrulewidth - \arraycolsep }
3311     \c_zero_dim
3312   }
3313   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3314 }
3315 \str_if_empty:NF \l_@@_name_str
3316 {
3317   \pgfnodealias
3318   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3319   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3320 }
3321 \endpgfpicture

3322 \bool_if:NT \g_@@_last_col_found_bool
3323 {
3324   \hbox_overlap_right:n
3325   {
3326     \skip_horizontal:N \g_@@_width_last_col_dim
3327     \bool_if:NT \l_@@_code_before_bool
3328     {
3329       \pgfsys@markposition
3330       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3331     }
3332     \pgfpicture
3333     \pgfrememberpicturepositiononpagetrue
3334     \pgfcoordinate
3335     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3336     \pgfpointorigin

```

```

3337         \str_if_empty:NF \l_@@_name_str
3338         {
3339             \pgfnodealias
3340             {
3341                 \l_@@_name_str - col
3342                 - \int_eval:n { \g_@@_col_total_int + 1 }
3343             }
3344             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3345         }
3346         \endpgfpicture
3347     }
3348 }
3349 \cr
3350 }

```

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

```

3351 \tl_const:Nn \c_@@_preamble_first_col_tl
3352 {
3353     >
3354     {

```

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).

```

3355         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3356         \bool_gset_true:N \g_@@_after_col_zero_bool
3357         \@@_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.

```

3358         \hbox_set:Nw \l_@@_cell_box
3359         \@@_math_toggle_token:
3360         \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”.

```

3361         \bool_lazy_and:nnT
3362         { \int_compare_p:nNn \c@iRow > 0 }
3363         {
3364             \bool_lazy_or_p:nn
3365             { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3366             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3367         }
3368         {
3369             \l_@@_code_for_first_col_tl
3370             \xglobal \colorlet { nicematrix-first-col } { . }
3371         }
3372     }

```

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`.

```

3373     l
3374     <
3375     {
3376         \@@_math_toggle_token:
3377         \hbox_set_end:
3378         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3379         \@@_adjust_size_box:
3380         \@@_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.

```

3381         \dim_gset:Nn \g_@@_width_first_col_dim
3382         { \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.

```

3383 \hbox_overlap_left:n
3384 {
3385   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3386     \@@_node_for_cell:
3387     { \box_use_drop:N \l_@@_cell_box }
3388     \skip_horizontal:N \l_@@_left_delim_dim
3389     \skip_horizontal:N \l_@@_left_margin_dim
3390     \skip_horizontal:N \l_@@_extra_left_margin_dim
3391   }
3392   \bool_gset_false:N \g_@@_empty_cell_bool
3393   \skip_horizontal:N -2\col@sep
3394 }
3395 }

```

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

```

3396 \tl_const:Nn \c_@@_preamble_last_col_tl
3397 {
3398   >
3399   {
3400     \bool_set_true:N \l_@@_in_last_col_bool

```

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

```

3401 \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.

```

3402 \bool_gset_true:N \g_@@_last_col_found_bool
3403 \int_gincr:N \c@jCol
3404 \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.

```

3405 \hbox_set:Nw \l_@@_cell_box
3406 \@@_math_toggle_token:
3407 \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”.

```

3408 \int_compare:nNnT \c@iRow > 0
3409 {
3410   \bool_lazy_or:nnT
3411     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3412     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3413     {
3414       \l_@@_code_for_last_col_tl
3415       \xglobal \colorlet { nicematrix-last-col } { . }
3416     }
3417 }
3418 }
3419 1
3420 <
3421 {
3422   \@@_math_toggle_token:
3423   \hbox_set_end:
3424   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3425   \@@_adjust_size_box:
3426   \@@_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.

```

3427 \dim_gset:Nn \g_@@_width_last_col_dim
3428   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3429 \skip_horizontal:N -2\col@sep

```

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

```

3430     \hbox_overlap_right:n
3431     {
3432         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3433         {
3434             \skip_horizontal:N \l_@@_right_delim_dim
3435             \skip_horizontal:N \l_@@_right_margin_dim
3436             \skip_horizontal:N \l_@@_extra_right_margin_dim
3437             \@@_node_for_cell:
3438         }
3439     }
3440     \bool_gset_false:N \g_@@_empty_cell_bool
3441 }
3442 }
```

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

```

3443 \NewDocumentEnvironment { NiceArray } { }
3444 {
3445     \bool_gset_false:N \g_@@_delims_bool
3446     \str_if_empty:NT \g_@@_name_env_str
3447     { \str_gset:Nn \g_@@_name_env_str { NiceArray } }
```

We put `.` and `.` for the delimiters but, in fact, that doesn't matter because these arguments won't be used in `{NiceArrayWithDelims}` (because the flag `\g_@@_delims_bool` is set to false).

```

3448     \NiceArrayWithDelims . .
3449 }
3450 { \endNiceArrayWithDelims }
```

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

```

3451 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3452 {
3453     \NewDocumentEnvironment { #1 NiceArray } { }
3454     {
3455         \bool_gset_true:N \g_@@_delims_bool
3456         \str_if_empty:NT \g_@@_name_env_str
3457         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3458         \@@_test_if_math_mode:
3459         \NiceArrayWithDelims #2 #3
3460     }
3461     { \endNiceArrayWithDelims }
3462 }

3463 \@@_def_env:nnn p ( )
3464 \@@_def_env:nnn b [ ]
3465 \@@_def_env:nnn B \{ \}
3466 \@@_def_env:nnn v | |
3467 \@@_def_env:nnn V \| \|
```

13 The environment `{NiceMatrix}` and its variants

```

3468 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3469 {
3470     \bool_set_false:N \l_@@_preamble_bool
3471     \tl_clear:N \l_tmpa_tl
3472     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3473     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3474     \tl_put_right:Nn \l_tmpa_tl
```

```

3475 {
3476   *
3477   {
3478     \int_case:nnF \l_@@_last_col_int
3479     {
3480       { -2 } { \c@MaxMatrixCols }
3481       { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3482     }
3483     { \int_eval:n { \l_@@_last_col_int - 1 } }
3484   }
3485   { #2 }
3486 }
3487 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3488 \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3489 }
3490 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3491 \clist_map_inline:nn { p , b , B , v , V }
3492 {
3493   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3494   {
3495     \bool_gset_true:N \g_@@_delims_bool
3496     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3497     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3498     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3499   }
3500   { \use:c { end #1 NiceArray } }
3501 }

```

We define also an environment {NiceMatrix}

```

3502 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3503 {
3504   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3505   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3506   \bool_lazy_or:nnT
3507     { \clist_if_empty_p:N \l_@@_vlines_clist }
3508     { \l_@@_except_borders_bool }
3509     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3510   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3511 }
3512 { \endNiceArray }

```

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

```

3513 \cs_new_protected:Npn \@@_NotEmpty:
3514 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3515 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3516 {

```

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.

```

3517   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3518     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3519   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3520   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3521   \tl_if_empty:NF \l_@@_short_caption_tl
3522   {
3523     \tl_if_empty:NT \l_@@_caption_tl

```

```

3524     {
3525         \@_error_or_warning:n { short-caption~without~caption }
3526         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3527     }
3528 }
3529 \tl_if_empty:NF \l_@@_label_tl
3530 {
3531     \tl_if_empty:NT \l_@@_caption_tl
3532     { \@_error_or_warning:n { label~without~caption } }
3533 }
3534 \NewDocumentEnvironment { TabularNote } { b }
3535 {
3536     \bool_if:NTF \l_@@_in_code_after_bool
3537     { \@_error_or_warning:n { TabularNote~in~CodeAfter } }
3538     {
3539         \tl_if_empty:NF \g_@@_tabularnote_tl
3540         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3541         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3542     }
3543 }
3544 { }
3545 \bool_set_true:N \l_@@_tabular_bool
3546 \NiceArray { #2 }
3547 }
3548 { \endNiceArray }

3549 \cs_set_protected:Npn \@_newcolumnmtype #1
3550 {
3551     \cs_if_free:cT { NC @ find @ #1 }
3552     { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3553     \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3554     \peek_meaning:NTF [
3555         { \newcol@ #1 }
3556         { \newcol@ #1 [ 0 ] }
3557     }

```

```

3558 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3559 {

```

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}`).

```

3560     \IfPackageLoadedTF { tabularx }
3561     { \newcolumnmtype { X } { \@_X } }
3562     { }
3563     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3564     \dim_zero_new:N \l_@@_width_dim
3565     \dim_set:Nn \l_@@_width_dim { #1 }
3566     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3567     \bool_set_true:N \l_@@_tabular_bool
3568     \NiceArray { #3 }
3569 }
3570 { \endNiceArray }

3571 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3572 {
3573     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3574     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3575     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3576     \bool_set_true:N \l_@@_tabular_bool
3577     \NiceArray { #3 }
3578 }
3579 { \endNiceArray }

```


15 After the construction of the array

```

3580 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3581 {
3582   \bool_lazy_all:nT
3583   {
3584     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3585     \l_@@_hvlines_bool
3586     { ! \g_@@_delims_bool }
3587     { ! \l_@@_except_borders_bool }
3588   }
3589   {
3590     \bool_set_true:N \l_@@_except_borders_bool
3591     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3592     {
3593       \_@@_stroke_block:nmn
3594       {
3595         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3596         draw = \l_@@_rules_color_tl
3597       }
3598       { 1-1 }
3599       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3600     }
3601   }
3602 }
3603 %
3604 %
3605 % \medskip
3606 % \begin{macrocode}
3607 \cs_new_protected:Npn \@@_after_array:
3608 {
3609   \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.

```

3610   \bool_if:NT \g_@@_last_col_found_bool
3611   { \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`.

```

3612   \bool_if:NT \l_@@_last_col_without_value_bool
3613   { \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.

```

3614   \bool_if:NT \l_@@_last_row_without_value_bool
3615   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3616   \tl_gput_right:Nx \g_@@_aux_tl
3617   {
3618     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3619     {
3620       \int_use:N \l_@@_first_row_int ,
3621       \int_use:N \c@iRow ,
3622       \int_use:N \g_@@_row_total_int ,
3623       \int_use:N \l_@@_first_col_int ,
3624       \int_use:N \c@jCol ,
3625       \int_use:N \g_@@_col_total_int
3626     }

```

```
3627 }
```

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`).

```
3628 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3629 {
3630   \tl_gput_right:Nx \g_@@_aux_tl
3631   {
3632     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3633     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3634   }
3635 }
3636 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3637 {
3638   \tl_gput_right:Nx \g_@@_aux_tl
3639   {
3640     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3641     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3642     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3643     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3644   }
3645 }
```

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

```
3646 \@@_create_diag_nodes:
```

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

```
3647 \pgfpicture
3648 \int_step_inline:nn \c@iRow
3649 {
3650   \pgfnodealias
3651   { \@@_env: - ##1 - last }
3652   { \@@_env: - ##1 - \int_use:N \c@jCol }
3653 }
3654 \int_step_inline:nn \c@jCol
3655 {
3656   \pgfnodealias
3657   { \@@_env: - last - ##1 }
3658   { \@@_env: - \int_use:N \c@iRow - ##1 }
3659 }
3660 \str_if_empty:NF \l_@@_name_str
3661 {
3662   \int_step_inline:nn \c@iRow
3663   {
3664     \pgfnodealias
3665     { \l_@@_name_str - ##1 - last }
3666     { \@@_env: - ##1 - \int_use:N \c@jCol }
3667   }
3668   \int_step_inline:nn \c@jCol
3669   {
3670     \pgfnodealias
3671     { \l_@@_name_str - last - ##1 }
3672     { \@@_env: - \int_use:N \c@iRow - ##1 }
3673   }
3674 }
3675 \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.

```
3676 \bool_if:NT \l_@@_parallelize_diags_bool
```

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

```

3677     {
3678         \int_gzero_new:N \g_@@_ddots_int
3679         \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.

```

3680         \dim_gzero_new:N \g_@@_delta_x_one_dim
3681         \dim_gzero_new:N \g_@@_delta_y_one_dim
3682         \dim_gzero_new:N \g_@@_delta_x_two_dim
3683         \dim_gzero_new:N \g_@@_delta_y_two_dim
3684     }
3685     \int_zero_new:N \l_@@_initial_i_int
3686     \int_zero_new:N \l_@@_initial_j_int
3687     \int_zero_new:N \l_@@_final_i_int
3688     \int_zero_new:N \l_@@_final_j_int
3689     \bool_set_false:N \l_@@_initial_open_bool
3690     \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.

```

3691     \bool_if:NT \l_@@_small_bool
3692     {
3693         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3694         \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.

```

3695         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3696         { 0.6 \l_@@_xdots_shorten_start_dim }
3697         \dim_set:Nn \l_@@_xdots_shorten_end_dim
3698         { 0.6 \l_@@_xdots_shorten_end_dim }
3699     }

```

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

```

3700     \@@_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.

```

3701     \@@_compute_corners:

```

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

```

3702     \@@_adjust_pos_of_blocks_seq:
3703     \@@_deal_with_rounded_corners:
3704     \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3705     \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3706     \IfPackageLoadedTF { tikz }
3707     {
3708         \tikzset
3709         {
3710             every~picture / .style =
3711             {
3712                 overlay ,
3713                 remember~picture ,
3714                 name~prefix = \@@_env: -
3715             }

```

```

3716     }
3717   }
3718   { }
3719   \cs_set_eq:NN \ialign \@@_old_ialign:
3720   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3721   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3722   \cs_set_eq:NN \OverBrace \@@_OverBrace
3723   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3724   \cs_set_eq:NN \line \@@_line
3725   \g_@@_pre_code_after_tl
3726   \tl_gclear:N \g_@@_pre_code_after_tl

```

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

```

3727   \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).

```

3728   \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`).

```

3729   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3730   { \@@_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:`.

```

3731   \bool_set_true:N \l_@@_in_code_after_bool
3732   \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3733   \scan_stop:
3734   \tl_gclear:N \g_nicematrix_code_after_tl
3735   \group_end:

```

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

```

3736   \tl_if_empty:NF \g_@@_pre_code_before_tl
3737   {
3738     \tl_gput_right:Nx \g_@@_aux_tl
3739     {
3740       \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3741       { \exp_not:V \g_@@_pre_code_before_tl }
3742     }
3743     \tl_gclear:N \g_@@_pre_code_before_tl
3744   }
3745   \tl_if_empty:NF \g_nicematrix_code_before_tl
3746   {
3747     \tl_gput_right:Nx \g_@@_aux_tl
3748     {
3749       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3750       { \exp_not:V \g_nicematrix_code_before_tl }
3751     }
3752     \tl_gclear:N \g_nicematrix_code_before_tl
3753   }

3754   \str_gclear:N \g_@@_name_env_str
3755   \@@_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

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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`.

```
3756 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3757 }
```

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`.

```
3758 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3759 { \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).

```
3760 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3761 {
3762   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3763   { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3764 }
```

The following command must *not* be protected.

```
3765 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3766 {
3767   { #1 }
3768   { #2 }
3769   {
3770     \int_compare:nNnTF { #3 } > { 99 }
3771     { \int_use:N \c@iRow }
3772     { #3 }
3773   }
3774   {
3775     \int_compare:nNnTF { #4 } > { 99 }
3776     { \int_use:N \c@jCol }
3777     { #4 }
3778   }
3779   { #5 }
3780 }
```

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).

```
3781 \hook_gput_code:nnn { begindocument } { . }
3782 {
3783   \cs_new_protected:Npx \@@_draw_dotted_lines:
3784   {
3785     \c_@@_pgfortikzpicture_tl
3786     \@@_draw_dotted_lines_i:
3787     \c_@@_endpgfortikzpicture_tl
3788   }
3789 }
```

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

```
3790 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3791 {
3792   \pgfrememberpicturepositiononpagetrue
3793   \pgf@relevantforpicturesizefalse
3794   \g_@@_HVdotsfor_lines_tl
3795   \g_@@_Vdots_lines_tl
```

```

3796 \g_@@_Ddots_lines_tl
3797 \g_@@_Iddots_lines_tl
3798 \g_@@_Cdots_lines_tl
3799 \g_@@_Ldots_lines_tl
3800 }

3801 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3802 {
3803   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3804   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3805 }

```

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

```

3806 \pgfdeclareshape { @@_diag_node }
3807 {
3808   \savedanchor { \five }
3809   {
3810     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3811     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3812   }
3813   \anchor { 5 } { \five }
3814   \anchor { center } { \pgfpointorigin }
3815 }

```

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).

```

3816 \cs_new_protected:Npn \@@_create_diag_nodes:
3817 {
3818   \pgfpicture
3819   \pgfrememberpicturepositiononpagetrue
3820   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3821   {
3822     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3823     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3824     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3825     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3826     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3827     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3828     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3829     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3830     \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 @a_diag_node) that we will construct.

```

3831     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3832     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3833     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3834     \str_if_empty:NF \l_@@_name_str
3835     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3836   }

```

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

```

3837   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3838   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3839   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3840   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3841   \pgfcoordinate
3842   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3843   \pgfnodealias
3844   { \@@_env: - last }
3845   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }

```

```

3846 \str_if_empty:NF \l_@@_name_str
3847 {
3848   \pgfnodealias
3849   { \l_@@_name_str - \int_use:N \l_tmpa_int }
3850   { \@@_env: - \int_use:N \l_tmpa_int }
3851   \pgfnodealias
3852   { \l_@@_name_str - last }
3853   { \@@_env: - last }
3854 }
3855 \endpgfpicture
3856 }

```

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.

```

3857 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3858 {

```

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

```

3859 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

3860 \int_set:Nn \l_@@_initial_i_int { #1 }
3861 \int_set:Nn \l_@@_initial_j_int { #2 }
3862 \int_set:Nn \l_@@_final_i_int { #1 }
3863 \int_set:Nn \l_@@_final_j_int { #2 }

```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```

3864 \bool_set_false:N \l_@@_stop_loop_bool
3865 \bool_do_until:Nn \l_@@_stop_loop_bool
3866 {
3867   \int_add:Nn \l_@@_final_i_int { #3 }
3868   \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3869     \bool_set_false:N \l_@@_final_open_bool
3870     \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3871     {
3872         \int_compare:nNnTF { #3 } = 1
3873         { \bool_set_true:N \l_@@_final_open_bool }
3874         {
3875             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3876             { \bool_set_true:N \l_@@_final_open_bool }
3877         }
3878     }
3879     {
3880         \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3881         {
3882             \int_compare:nNnT { #4 } = { -1 }
3883             { \bool_set_true:N \l_@@_final_open_bool }
3884         }
3885         {
3886             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3887             {
3888                 \int_compare:nNnT { #4 } = 1
3889                 { \bool_set_true:N \l_@@_final_open_bool }
3890             }
3891         }
3892     }
3893     \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.

```

3894     {

```

We do a step backwards.

```

3895         \int_sub:Nn \l_@@_final_i_int { #3 }
3896         \int_sub:Nn \l_@@_final_j_int { #4 }
3897         \bool_set_true:N \l_@@_stop_loop_bool
3898     }

```

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`.

```

3899     {
3900         \cs_if_exist:cTF
3901         {
3902             @@ _ dotted _
3903             \int_use:N \l_@@_final_i_int -
3904             \int_use:N \l_@@_final_j_int
3905         }
3906         {
3907             \int_sub:Nn \l_@@_final_i_int { #3 }
3908             \int_sub:Nn \l_@@_final_j_int { #4 }
3909             \bool_set_true:N \l_@@_final_open_bool
3910             \bool_set_true:N \l_@@_stop_loop_bool
3911         }
3912         {
3913             \cs_if_exist:cTF
3914             {
3915                 pgf @ sh @ ns @ \@@_env:
3916                 - \int_use:N \l_@@_final_i_int
3917                 - \int_use:N \l_@@_final_j_int
3918             }
3919             { \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.


```

3920         {
3921             \cs_set:cpn
3922             {
3923                 @@ _ dotted _
3924                 \int_use:N \l_@@_final_i_int -
3925                 \int_use:N \l_@@_final_j_int
3926             }
3927             { }
3928         }
3929     }
3930 }
3931 }

```

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

```

3932 \bool_set_false:N \l_@@_stop_loop_bool
3933 \bool_do_until:Nn \l_@@_stop_loop_bool
3934 {
3935     \int_sub:Nn \l_@@_initial_i_int { #3 }
3936     \int_sub:Nn \l_@@_initial_j_int { #4 }
3937     \bool_set_false:N \l_@@_initial_open_bool
3938     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3939     {
3940         \int_compare:nNnTF { #3 } = 1
3941         { \bool_set_true:N \l_@@_initial_open_bool }
3942         {
3943             \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3944             { \bool_set_true:N \l_@@_initial_open_bool }
3945         }
3946     }
3947     {
3948         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3949         {
3950             \int_compare:nNnT { #4 } = 1
3951             { \bool_set_true:N \l_@@_initial_open_bool }
3952         }
3953         {
3954             \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3955             {
3956                 \int_compare:nNnT { #4 } = { -1 }
3957                 { \bool_set_true:N \l_@@_initial_open_bool }
3958             }
3959         }
3960     }
3961     \bool_if:NTF \l_@@_initial_open_bool
3962     {
3963         \int_add:Nn \l_@@_initial_i_int { #3 }
3964         \int_add:Nn \l_@@_initial_j_int { #4 }
3965         \bool_set_true:N \l_@@_stop_loop_bool
3966     }
3967     {
3968         \cs_if_exist:cTF
3969         {
3970             @@ _ dotted _
3971             \int_use:N \l_@@_initial_i_int -
3972             \int_use:N \l_@@_initial_j_int
3973         }
3974         {
3975             \int_add:Nn \l_@@_initial_i_int { #3 }
3976             \int_add:Nn \l_@@_initial_j_int { #4 }
3977             \bool_set_true:N \l_@@_initial_open_bool
3978             \bool_set_true:N \l_@@_stop_loop_bool

```

```

3979     }
3980     {
3981         \cs_if_exist:cTF
3982         {
3983             pgf @ sh @ ns @ \@@_env:
3984             - \int_use:N \l_@@_initial_i_int
3985             - \int_use:N \l_@@_initial_j_int
3986         }
3987         { \bool_set_true:N \l_@@_stop_loop_bool }
3988         {
3989             \cs_set:cpn
3990             {
3991                 @@ _ dotted _
3992                 \int_use:N \l_@@_initial_i_int -
3993                 \int_use:N \l_@@_initial_j_int
3994             }
3995             { }
3996         }
3997     }
3998 }
3999 }

```

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.

```

4000     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4001     {
4002         { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4003         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4004         { \int_use:N \l_@@_final_i_int }
4005         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4006         { } % for the name of the block
4007     }
4008 }

```

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).

```

4009 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4010 {
4011     \int_set:Nn \l_@@_row_min_int 1
4012     \int_set:Nn \l_@@_col_min_int 1
4013     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4014     \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`.

```

4015     \seq_map_inline:Nn \g_@@_submatrix_seq
4016     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4017 }

```

#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.

```

4018 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4019 {
4020     \bool_if:nT
4021     {
4022         \int_compare_p:n { #3 <= #1 }
4023         && \int_compare_p:n { #1 <= #5 }
4024         && \int_compare_p:n { #4 <= #2 }

```

```

4025     && \int_compare_p:n { #2 <= #6 }
4026   }
4027   {
4028     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4029     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4030     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4031     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4032   }
4033 }

4034 \cs_new_protected:Npn \@@_set_initial_coords:
4035 {
4036   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4037   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4038 }
4039 \cs_new_protected:Npn \@@_set_final_coords:
4040 {
4041   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4042   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4043 }
4044 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4045 {
4046   \pgfpointanchor
4047   {
4048     \@@_env:
4049     - \int_use:N \l_@@_initial_i_int
4050     - \int_use:N \l_@@_initial_j_int
4051   }
4052   { #1 }
4053   \@@_set_initial_coords:
4054 }
4055 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4056 {
4057   \pgfpointanchor
4058   {
4059     \@@_env:
4060     - \int_use:N \l_@@_final_i_int
4061     - \int_use:N \l_@@_final_j_int
4062   }
4063   { #1 }
4064   \@@_set_final_coords:
4065 }

4066 \cs_new_protected:Npn \@@_open_x_initial_dim:
4067 {
4068   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4069   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4070   {
4071     \cs_if_exist:cT
4072     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4073     {
4074       \pgfpointanchor
4075       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4076       { west }
4077       \dim_set:Nn \l_@@_x_initial_dim
4078       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4079     }
4080   }

```

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

```

4081   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4082   {
4083     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4084     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

4085     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4086   }
4087 }
4088 \cs_new_protected:Npn \@@_open_x_final_dim:
4089 {
4090   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4091   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4092   {
4093     \cs_if_exist:cT
4094     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4095     {
4096       \pgfpointanchor
4097       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4098       { east }
4099       \dim_set:Nn \l_@@_x_final_dim
4100       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4101     }
4102   }

```

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

```

4103   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4104   {
4105     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4106     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4107     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4108   }
4109 }

```

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.

```

4110 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4111 {
4112   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4113   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4114   {
4115     \@@_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.

```

4116   \group_begin:
4117   \int_compare:nNnTF { #1 } = 0
4118   { \color { nicematrix-first-row } }
4119   {

```

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.

```

4120     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4121     { \color { nicematrix-last-row } }
4122   }
4123   \keys_set:nn { NiceMatrix / xdots } { #3 }
4124   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4125   \@@_actually_draw_Ldots:
4126   \group_end:
4127 }
4128 }

```

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`.

```

4129 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4130 {
4131   \bool_if:NTF \l_@@_initial_open_bool
4132   {
4133     \@@_open_x_initial_dim:
4134     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4135     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4136   }
4137   { \@@_set_initial_coords_from_anchor:n { base~east } }
4138   \bool_if:NTF \l_@@_final_open_bool
4139   {
4140     \@@_open_x_final_dim:
4141     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4142     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4143   }
4144   { \@@_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 (?).

```

4145   \dim_add:NN \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4146   \dim_add:NN \l_@@_y_final_dim \l_@@_xdots_radius_dim
4147   \@@_draw_line:
4148 }

```

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.

```

4149 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4150 {
4151   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4152   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4153   {
4154     \@@_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.

```

4155   \group_begin:
4156   \int_compare:nNnTF { #1 } = 0
4157   { \color { nicematrix-first-row } }
4158   {

```

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.

```

4159     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4160     { \color { nicematrix-last-row } }
4161   }
4162   \keys_set:nn { NiceMatrix / xdots } { #3 }
4163   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4164   \@@_actually_draw_Cdots:
4165   \group_end:
4166 }
4167 }

```

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.

```

4168 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4169 {
4170   \bool_if:NTF \l_@@_initial_open_bool
4171     { \@@_open_x_initial_dim: }
4172     { \@@_set_initial_coords_from_anchor:n { mid~east } }
4173   \bool_if:NTF \l_@@_final_open_bool
4174     { \@@_open_x_final_dim: }
4175     { \@@_set_final_coords_from_anchor:n { mid~west } }
4176   \bool_lazy_and:nnTF
4177     \l_@@_initial_open_bool
4178     \l_@@_final_open_bool
4179   {
4180     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4181     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4182     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4183     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4184     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4185   }
4186   {
4187     \bool_if:NT \l_@@_initial_open_bool
4188       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4189     \bool_if:NT \l_@@_final_open_bool
4190       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4191   }
4192   \@@_draw_line:
4193 }
4194 \cs_new_protected:Npn \@@_open_y_initial_dim:
4195 {
4196   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4197   \dim_set:Nn \l_@@_y_initial_dim
4198   {
4199     \fp_to_dim:n
4200     {
4201       \pgf@y
4202       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4203     }
4204   } % modified 6.13c
4205   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4206   {
4207     \cs_if_exist:cT
4208     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4209     {
4210       \pgfpointanchor
4211       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4212       { north }
4213       \dim_set:Nn \l_@@_y_initial_dim
4214       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4215     }
4216   }
4217 }
4218 \cs_new_protected:Npn \@@_open_y_final_dim:
4219 {
4220   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4221   \dim_set:Nn \l_@@_y_final_dim
4222   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4223   % modified 6.13c

```

```

4224 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4225 {
4226   \cs_if_exist:cT
4227   { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4228   {
4229     \pgfpointanchor
4230     { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4231     { south }
4232     \dim_set:Nn \l_@@_y_final_dim
4233     { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4234   }
4235 }
4236 }

```

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.

```

4237 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4238 {
4239   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4240   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4241   {
4242     \@@_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.

```

4243   \group_begin:
4244     \int_compare:nNnTF { #2 } = 0
4245     { \color { nicematrix-first-col } }
4246     {
4247       \int_compare:nNnT { #2 } = \l_@@_last_col_int
4248       { \color { nicematrix-last-col } }
4249     }
4250     \keys_set:nn { NiceMatrix / xdots } { #3 }
4251     \tl_if_empty:VF \l_@@_xdots_color_tl
4252     { \color { \l_@@_xdots_color_tl } }
4253     \@@_actually_draw_Vdots:
4254   \group_end:
4255 }
4256 }

```

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`.

```

4257 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4258 {

```

The boolean `\l_tmpa_bool` indicates whether the column is of type 1 or may be considered as if.

```

4259   \bool_set_false:N \l_tmpa_bool

```

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

```

4260   \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
4261   {
4262     \@@_set_initial_coords_from_anchor:n { south-west }
4263     \@@_set_final_coords_from_anchor:n { north-west }

```

```

4264     \bool_set:Nn \l_tmpa_bool
4265     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4266 }

```

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

```

4267 \bool_if:NTF \l_@@_initial_open_bool
4268 \@@_open_y_initial_dim:
4269 { \@@_set_initial_coords_from_anchor:n { south } }
4270 \bool_if:NTF \l_@@_final_open_bool
4271 \@@_open_y_final_dim:
4272 { \@@_set_final_coords_from_anchor:n { north } }
4273 \bool_if:NTF \l_@@_initial_open_bool
4274 {
4275   \bool_if:NTF \l_@@_final_open_bool
4276   {
4277     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4278     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4279     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4280     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4281     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim

```

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. That's why we do an adjustment for that case.

```

4282 \int_compare:nNnT \l_@@_last_col_int > { -2 }
4283 {
4284   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
4285   {
4286     \dim_set:Nn \l_tmpa_dim
4287     {
4288       \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim
4289       + 2 mm
4290     }
4291     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4292     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
4293   }
4294 }
4295 }
4296 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
4297 }
4298 {
4299   \bool_if:NTF \l_@@_final_open_bool
4300   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
4301   {

```

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.

```

4302 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4303 {
4304   \dim_set:Nn \l_@@_x_initial_dim
4305   {
4306     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4307     \l_@@_x_initial_dim \l_@@_x_final_dim
4308   }
4309   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4310 }
4311 }
4312 }
4313 \@@_draw_line:
4314 }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonal 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.

```

4315 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4316 {
4317   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4318   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4319   {
4320     \@@_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.

```

4321     \group_begin:
4322     \keys_set:nn { NiceMatrix / xdots } { #3 }
4323     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4324     \@@_actually_draw_Ddots:
4325   \group_end:
4326 }
4327 }

```

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`.

```

4328 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4329 {
4330   \bool_if:NTF \l_@@_initial_open_bool
4331   {
4332     \@@_open_y_initial_dim:
4333     \@@_open_x_initial_dim:
4334   }
4335   { \@@_set_initial_coords_from_anchor:n { south-east } }
4336   \bool_if:NTF \l_@@_final_open_bool
4337   {
4338     \@@_open_x_final_dim:
4339     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4340   }
4341   { \@@_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.

```

4342   \bool_if:NT \l_@@_parallelize_diags_bool
4343   {
4344     \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).

```

4345     \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.

```

4346   {
4347     \dim_gset:Nn \g_@@_delta_x_one_dim
4348     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4349     \dim_gset:Nn \g_@@_delta_y_one_dim
4350     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4351   }

```

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`.

```

4352     {
4353         \dim_set:Nn \l_@@_y_final_dim
4354         {
4355             \l_@@_y_initial_dim +
4356             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4357             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4358         }
4359     }
4360 }
4361 \@@_draw_line:
4362 }

```

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.

```

4363 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4364 {
4365     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4366     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4367     {
4368         \@@_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.

```

4369     \group_begin:
4370     \keys_set:nn { NiceMatrix / xdots } { #3 }
4371     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4372     \@@_actually_draw_Iddots:
4373     \group_end:
4374 }
4375 }

```

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`.

```

4376 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4377 {
4378     \bool_if:NTF \l_@@_initial_open_bool
4379     {
4380         \@@_open_y_initial_dim:
4381         \@@_open_x_initial_dim:
4382     }
4383     { \@@_set_initial_coords_from_anchor:n { south-west } }
4384     \bool_if:NTF \l_@@_final_open_bool
4385     {
4386         \@@_open_y_final_dim:
4387         \@@_open_x_final_dim:
4388     }
4389     { \@@_set_final_coords_from_anchor:n { north-east } }
4390     \bool_if:NT \l_@@_parallelize_diags_bool
4391     {

```

```

4392 \int_gincr:N \g_@@_iddots_int
4393 \int_compare:nNnTF \g_@@_iddots_int = 1
4394 {
4395     \dim_gset:Nn \g_@@_delta_x_two_dim
4396     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4397     \dim_gset:Nn \g_@@_delta_y_two_dim
4398     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4399 }
4400 {
4401     \dim_set:Nn \l_@@_y_final_dim
4402     {
4403         \l_@@_y_initial_dim +
4404         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4405         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4406     }
4407 }
4408 }
4409 \@@_draw_line:
4410 }

```

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`

```

4411 \cs_new_protected:Npn \@@_draw_line:
4412 {
4413     \pgfrememberpicturerepositiononpagetrue
4414     \pgf@relevantforpicturesizefalse
4415     \bool_lazy_or:nnTF
4416     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4417     \l_@@_dotted_bool
4418     \@@_draw_standard_dotted_line:
4419     \@@_draw_unstandard_dotted_line:
4420 }

```

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.

```

4421 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4422 {
4423     \begin { scope }
4424     \@@_draw_unstandard_dotted_line:o
4425     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4426 }

```

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.

```

4427 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4428 {
4429   \@@_draw_unstandard_dotted_line:nVV
4430   { #1 }
4431   \l_@@_xdots_up_tl
4432   \l_@@_xdots_down_tl
4433 }
4434 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4435 \hook_gput_code:nnn { begindocument } { . }
4436 {
4437   \IfPackageLoadedTF { tikz }
4438   {
4439     \tikzset
4440     {
4441       @@_node_above / .style = { sloped , above } ,
4442       @@_node_below / .style = { sloped , below }
4443     }
4444   }
4445   { }
4446 }

4447 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
4448 {
4449   \bool_if:NT \l_@@_xdots_h_labels_bool
4450   {
4451     \tikzset
4452     {
4453       @@_node_above / .style = { auto = left } ,
4454       @@_node_below / .style = { auto = right }
4455     }
4456   }
4457   \draw
4458   [
4459     shorten-> = \l_@@_xdots_shorten_end_dim ,
4460     shorten-< = \l_@@_xdots_shorten_start_dim ,
4461     #1
4462   ]
4463   ( \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).

```

4464     -- node [ @@_node_above ] { $ \scriptstyle #2 $ }
4465     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4466     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4467   \end { scope }
4468 }
4469 \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).

```

4470 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4471 {
4472   \bool_lazy_and:nnF
4473   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4474   { \tl_if_empty_p:N \l_@@_xdots_down_tl }

```

```

4475 {
4476   \pgfscope
4477   \pgftransformshift
4478   {
4479     \pgfpointlineatime { 0.5 }
4480     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4481     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4482   }
4483   \fp_set:Nn \l_tmpa_fp
4484   {
4485     atand
4486     (
4487       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4488       \l_@@_x_final_dim - \l_@@_x_initial_dim
4489     )
4490   }
4491   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4492   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4493   \pgfnode
4494   { rectangle }
4495   { south }
4496   {
4497     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4498     {
4499       \c_math_toggle_token
4500       \scriptstyle \l_@@_xdots_up_tl
4501       \c_math_toggle_token
4502     }
4503   }
4504   { }
4505   { \pgfusepath { } }
4506   \pgfnode
4507   { rectangle }
4508   { north }
4509   {
4510     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4511     {
4512       \c_math_toggle_token
4513       \scriptstyle \l_@@_xdots_down_tl
4514       \c_math_toggle_token
4515     }
4516   }
4517   { }
4518   { \pgfusepath { } }
4519   \endpgfscope
4520 }
4521 \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.

```

4522 \dim_zero_new:N \l_@@_l_dim
4523 \dim_set:Nn \l_@@_l_dim
4524 {
4525   \fp_to_dim:n
4526   {
4527     sqrt
4528     (
4529       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4530       +
4531       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4532     )
4533   }
4534 }

```

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.

```

4535     \bool_lazy_or:nnF
4536     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4537     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4538     \@@_draw_standard_dotted_line_i:
4539     \group_end:
4540 }
4541 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4542 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4543 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

4544     \bool_if:NTF \l_@@_initial_open_bool
4545     {
4546         \bool_if:NTF \l_@@_final_open_bool
4547         {
4548             \int_set:Nn \l_tmpa_int
4549             { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
4550         }
4551         {
4552             \int_set:Nn \l_tmpa_int
4553             {
4554                 \dim_ratio:nn
4555                 { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
4556                 \l_@@_xdots_inter_dim
4557             }
4558         }
4559     }
4560     {
4561         \bool_if:NTF \l_@@_final_open_bool
4562         {
4563             \int_set:Nn \l_tmpa_int
4564             {
4565                 \dim_ratio:nn
4566                 { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
4567                 \l_@@_xdots_inter_dim
4568             }
4569         }
4570         {
4571             \int_set:Nn \l_tmpa_int
4572             {
4573                 \dim_ratio:nn
4574                 {
4575                     \l_@@_l_dim
4576                     - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4577                 }
4578                 \l_@@_xdots_inter_dim
4579             }
4580         }
4581     }

```

The dimensions $\l_1\text{tmpa_dim}$ and $\l_1\text{tmpb_dim}$ are the coordinates of the vector between two dots in the dotted line.

```

4582     \dim_set:Nn \l_tmpa_dim
4583     {
4584         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4585         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4586     }
4587     \dim_set:Nn \l_tmpb_dim
4588     {

```

```

4589      ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4590      \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4591    }

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.

4592    \dim_gadd:Nn \l_@@_x_initial_dim
4593    {
4594      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4595      \dim_ratio:nn
4596      {
4597        \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4598        + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4599      }
4600      { 2 \l_@@_l_dim }
4601    }
4602    \dim_gadd:Nn \l_@@_y_initial_dim
4603    {
4604      ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4605      \dim_ratio:nn
4606      {
4607        \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4608        + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4609      }
4610      { 2 \l_@@_l_dim }
4611    }
4612    \pgf@relevantforpicturesizefalse
4613    \int_step_inline:nnn 0 \l_tmpa_int
4614    {
4615      \pgfpathcircle
4616      { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4617      { \l_@@_xdots_radius_dim }
4618      \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4619      \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4620    }
4621    \pgfusepathqfill
4622  }

```

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.

```

4623 \hook_gput_code:nnn { begindocument } { . }
4624 {
4625   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
4626   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4627   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4628   {
4629     \int_compare:nNnTF \c@jCol = 0
4630     { \@@_error:nn { in~first~col } \Ldots }
4631     {
4632       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int

```

```

4633         { \@@_error:nn { in~last~col } \Ldots }
4634     {
4635         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4636         { #1 , down = #2 , up = #3 }
4637     }
4638 }
4639 \bool_if:NF \l_@@_nullify_dots_bool
4640 { \phantom { \ensuremath { \@@_old_ldots } } }
4641 \bool_gset_true:N \g_@@_empty_cell_bool
4642 }

\exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
{
4643     \int_compare:nNnTF \c@jCol = 0
4644     { \@@_error:nn { in~first~col } \Cdots }
4645     {
4646         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4647         { \@@_error:nn { in~last~col } \Cdots }
4648         {
4649             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4650             { #1 , down = #2 , up = #3 }
4651         }
4652     }
4653 }
4654 \bool_if:NF \l_@@_nullify_dots_bool
4655 { \phantom { \ensuremath { \@@_old_cdots } } }
4656 \bool_gset_true:N \g_@@_empty_cell_bool
4657 }
4658 }

\exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
{
4659     \int_compare:nNnTF \c@iRow = 0
4660     { \@@_error:nn { in~first~row } \Vdots }
4661     {
4662         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4663         { \@@_error:nn { in~last~row } \Vdots }
4664         {
4665             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4666             { #1 , down = #2 , up = #3 }
4667         }
4668     }
4669 }
4670 \bool_if:NF \l_@@_nullify_dots_bool
4671 { \phantom { \ensuremath { \@@_old_vdots } } }
4672 \bool_gset_true:N \g_@@_empty_cell_bool
4673 }
4674 }

\exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
{
4675     \int_case:nnF \c@iRow
4676     {
4677         0 { \@@_error:nn { in~first~row } \Ddots }
4678         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4679     }
4680     {
4681         \int_case:nnF \c@jCol
4682         {
4683             0 { \@@_error:nn { in~first~col } \Ddots }
4684             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4685         }
4686         {
4687             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4688             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4689         }
4690     }

```



```

4691         { #1 , down = #2 , up = #3 }
4692     }
4693
4694 }
4695 \bool_if:NF \l_@@_nullify_dots_bool
4696 { \phantom { \ensuremath { \@@_old_ddots } } }
4697 \bool_gset_true:N \g_@@_empty_cell_bool
4698 }

```



```

4699 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4700 {
4701     \int_case:nnF \c@iRow
4702     {
4703         0 { \@@_error:nn { in~first~row } \Iddots }
4704         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4705     }
4706     {
4707         \int_case:nnF \c@jCol
4708         {
4709             0 { \@@_error:nn { in~first~col } \Iddots }
4710             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4711         }
4712         {
4713             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4714             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4715             { #1 , down = #2 , up = #3 }
4716         }
4717     }
4718     \bool_if:NF \l_@@_nullify_dots_bool
4719     { \phantom { \ensuremath { \@@_old_iddots } } }
4720     \bool_gset_true:N \g_@@_empty_cell_bool
4721 }
4722 }

```

End of the \AddToHook.

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

```

4723 \keys_define:nn { NiceMatrix / Ddots }
4724 {
4725     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4726     draw-first .default:n = true ,
4727     draw-first .value_forbidden:n = true
4728 }

```

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

```

4729 \cs_new_protected:Npn \@@_Hspace:
4730 {
4731     \bool_gset_true:N \g_@@_empty_cell_bool
4732     \hspace
4733 }

```

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`.

```

4734 \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`.

```

4735 \cs_new:Npn \@@_Hdotsfor:
4736 {

```

```

4737 \bool_lazy_and:nnTF
4738 { \int_compare_p:nNn \c@jCol = 0 }
4739 { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4740 {
4741   \bool_if:NTF \g_@@_after_col_zero_bool
4742   {
4743     \multicolumn { 1 } { c } { }
4744     \@@_Hdotsfor_i
4745   }
4746   { \@@_fatal:n { Hdotsfor~in~col~0 } }
4747 }
4748 {
4749   \multicolumn { 1 } { c } { }
4750   \@@_Hdotsfor_i
4751 }
4752 }

```

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:`).

```

4753 \hook_gput_code:nnn { begindocument } { . }
4754 {
4755   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4756   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

4757 \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4758 {
4759   \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4760   {
4761     \@@_Hdotsfor:nnnn
4762     { \int_use:N \c@iRow }
4763     { \int_use:N \c@jCol }
4764     { #2 }
4765     {
4766       #1 , #3 ,
4767       down = \exp_not:n { #4 } ,
4768       up = \exp_not:n { #5 }
4769     }
4770   }
4771   \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4772 }
4773 }

```

```

4774 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4775 {
4776   \bool_set_false:N \l_@@_initial_open_bool
4777   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4778 \int_set:Nn \l_@@_initial_i_int { #1 }
4779 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4780 \int_compare:nNnTF { #2 } = 1
4781 {
4782   \int_set:Nn \l_@@_initial_j_int 1
4783   \bool_set_true:N \l_@@_initial_open_bool
4784 }
4785 {
4786   \cs_if_exist:cTF
4787   {
4788     pgf @ sh @ ns @ \@@_env:
4789     - \int_use:N \l_@@_initial_i_int

```

```

4790         - \int_eval:n { #2 - 1 }
4791     }
4792     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4793     {
4794         \int_set:Nn \l_@@_initial_j_int { #2 }
4795         \bool_set_true:N \l_@@_initial_open_bool
4796     }
4797 }
4798 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4799 {
4800     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4801     \bool_set_true:N \l_@@_final_open_bool
4802 }
4803 {
4804     \cs_if_exist:cTF
4805     {
4806         pgf @ sh @ ns @ \@@_env:
4807         - \int_use:N \l_@@_final_i_int
4808         - \int_eval:n { #2 + #3 }
4809     }
4810     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4811     {
4812         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4813         \bool_set_true:N \l_@@_final_open_bool
4814     }
4815 }
4816 \group_begin:
4817 \int_compare:nNnTF { #1 } = 0
4818 { \color { nicematrix-first-row } }
4819 {
4820     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4821     { \color { nicematrix-last-row } }
4822 }
4823 \keys_set:nn { NiceMatrix / xdots } { #4 }
4824 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4825 \@@_actually_draw_Ldots:
4826 \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).

```

4827     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4828     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4829 }

4830 \hook_gput_code:nnn { begindocument } { . }
4831 {
4832     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4833     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4834     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4835     {
4836         \bool_gset_true:N \g_@@_empty_cell_bool % added
4837         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4838         {
4839             \@@_Vdotsfor:nnnn
4840             { \int_use:N \c@iRow }
4841             { \int_use:N \c@jCol }
4842             { #2 }
4843             {
4844                 #1 , #3 ,
4845                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4846             }

```

```

4847     }
4848   }
4849 }

```

Enf of \AddToHook.

```

4850 \cs_new_protected:Npn \l_@@_Vdotsfor:nnnn #1 #2 #3 #4
4851 {
4852   \bool_set_false:N \l_@@_initial_open_bool
4853   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4854   \int_set:Nn \l_@@_initial_j_int { #2 }
4855   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4856   \int_compare:nNnTF { #1 } = 1
4857   {
4858     \int_set:Nn \l_@@_initial_i_int 1
4859     \bool_set_true:N \l_@@_initial_open_bool
4860   }
4861   {
4862     \cs_if_exist:cTF
4863     {
4864       pgf @ sh @ ns @ \l_@@_env:
4865       - \int_eval:n { #1 - 1 }
4866       - \int_use:N \l_@@_initial_j_int
4867     }
4868     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4869     {
4870       \int_set:Nn \l_@@_initial_i_int { #1 }
4871       \bool_set_true:N \l_@@_initial_open_bool
4872     }
4873   }
4874   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4875   {
4876     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4877     \bool_set_true:N \l_@@_final_open_bool
4878   }
4879   {
4880     \cs_if_exist:cTF
4881     {
4882       pgf @ sh @ ns @ \l_@@_env:
4883       - \int_eval:n { #1 + #3 }
4884       - \int_use:N \l_@@_final_j_int
4885     }
4886     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4887     {
4888       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4889       \bool_set_true:N \l_@@_final_open_bool
4890     }
4891   }
4892   \group_begin:
4893   \int_compare:nNnTF { #2 } = 0
4894   { \color { nicematrix-first-col } }
4895   {
4896     \int_compare:nNnT { #2 } = \g_@@_col_total_int
4897     { \color { nicematrix-last-col } }
4898   }
4899   \keys_set:nn { NiceMatrix / xdots } { #4 }
4900   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4901   \@@_actually_draw_Vdots:
4902   \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).

```

4903   \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4904   { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4905 }

```

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

```

4906 \NewDocumentCommand \@@_rotate: { 0 { } }
4907 {
4908   \bool_gset_true:N \g_@@_rotate_bool
4909   \keys_set:nn { NiceMatrix / rotate } { #1 }
4910 }

4911 \keys_define:nn { NiceMatrix / rotate }
4912 {
4913   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
4914   c .value_forbidden:n = true ,
4915   unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
4916 }

```

19 The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format i - j , our command applies the command `\int_eval:n` to i and j ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹³

```

4917 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4918 {
4919   \tl_if_empty:nTF { #2 }
4920   { #1 }
4921   { \@@_double_int_eval_i:n #1-#2 \q_stop }
4922 }
4923 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
4924 { \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).

```

4925 \hook_gput_code:nnn { begindocument } { . }
4926 {
4927   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4928   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4929   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl

```

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

```

4930 {
4931   \group_begin:
4932   \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4933   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4934   \use:e
4935   {
4936     \@@_line_i:nn
4937     { \@@_double_int_eval:n #2 - \q_stop }
4938     { \@@_double_int_eval:n #3 - \q_stop }
4939   }
4940   \group_end:
4941 }
4942 }
4943 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4944 {
4945   \bool_set_false:N \l_@@_initial_open_bool
4946   \bool_set_false:N \l_@@_final_open_bool
4947   \bool_if:nTF
4948   {
4949     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4950     ||
4951     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4952   }
4953   {
4954     \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4955   }

```

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

```

4956   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
4957 }
4958 \hook_gput_code:nnn { begindocument } { . }
4959 {
4960   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4961   {

```

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:`.

```

4962   \c_@@_pgfortikzpicture_tl
4963   \@@_draw_line_iii:nn { #1 } { #2 }
4964   \c_@@_endpgfortikzpicture_tl
4965 }
4966 }

```

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

```

4967 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4968 {
4969   \pgfrememberpicturepositiononpagetrue
4970   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4971   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4972   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4973   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4974   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4975   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4976   \@@_draw_line:
4977 }

```

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

```

4978 \keys_define:nn { NiceMatrix / RowStyle }
4979 {
4980   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4981   cell-space-top-limit .initial:n = \c_zero_dim ,
4982   cell-space-top-limit .value_required:n = true ,
4983   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4984   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4985   cell-space-bottom-limit .value_required:n = true ,
4986   cell-space-limits .meta:n =
4987     {
4988       cell-space-top-limit = #1 ,
4989       cell-space-bottom-limit = #1 ,
4990     } ,
4991   color .tl_set:N = \l_@@_color_tl ,
4992   color .value_required:n = true ,
4993   bold .bool_set:N = \l_tmpa_bool ,
4994   bold .default:n = true ,
4995   bold .initial:n = false ,
4996   nb-rows .code:n =
4997     \str_if_eq:nnTF { #1 } { * }
4998     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
4999     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5000   nb-rows .value_required:n = true ,
5001   rowcolor .tl_set:N = \l_tmpa_tl ,
5002   rowcolor .value_required:n = true ,
5003   rowcolor .initial:n = ,
5004   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
5005 }

```

```

5006 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5007 {
5008   \group_begin:
5009   \tl_clear:N \l_tmpa_tl % value of \rowcolor
5010   \tl_clear:N \l_@@_color_tl
5011   \int_set:Nn \l_@@_key_nb_rows_int 1
5012   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5013   \tl_if_empty:NF \l_tmpa_tl
5014   {

```

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

```

5015     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5016     {

```

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

```

5017     \@@_exp_color_arg:N \@@_rectanglecolor \l_tmpa_tl
5018     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5019     { \int_use:N \c@iRow - * }
5020   }

```

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

```

5021     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
5022     {
5023       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5024       {
5025         \@@_exp_color_arg:N \@@_rowcolor \l_tmpa_tl
5026         {
5027           \int_eval:n { \c@iRow + 1 }
5028           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5029         }
5030       }

```

```

5031     }
5032   }
5033   \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
5034   \tl_gput_right:Nx \g_@@_row_style_tl
5035   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5036   \tl_gput_right:Nn \g_@@_row_style_tl { #2 }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
5037   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5038   {
5039     \tl_gput_right:Nx \g_@@_row_style_tl
5040     {
5041       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5042       {
5043         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5044         { \dim_use:N \l_tmpa_dim }
5045       }
5046     }
5047   }
\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
5048   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5049   {
5050     \tl_gput_right:Nx \g_@@_row_style_tl
5051     {
5052       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5053       {
5054         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5055         { \dim_use:N \l_tmpb_dim }
5056       }
5057     }
5058   }
\l_@@_color_tl is the value of the key color of \RowStyle.
5059   \tl_if_empty:NF \l_@@_color_tl
5060   {
5061     \tl_gput_right:Nx \g_@@_row_style_tl
5062     {
5063       \mode_leave_vertical:
5064       \@@_color:n { \l_@@_color_tl }
5065     }
5066   }
\l_tmpa_bool is the value of the key bold.
5067   \bool_if:NT \l_tmpa_bool
5068   {
5069     \tl_gput_right:Nn \g_@@_row_style_tl
5070     {
5071       \if_mode_math:
5072         \c_math_toggle_token
5073         \bfseries \boldmath
5074         \c_math_toggle_token
5075       \else:
5076         \bfseries \boldmath
5077       \fi:
5078     }
5079   }
5080   \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5081   \group_end:
5082   \g_@@_row_style_tl
5083   \ignorespaces
5084 }

```


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).

```
5085 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5086 {
```

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.

```
5087 \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`.

```
5088 \str_if_in:nnF { #1 } { !! }
5089 {
5090   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5091   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5092 }
5093 \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

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

```
5094 {
5095   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5096   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5097 }
```

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

```
5098 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5099 }
```

```
5100 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5101 \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`).

```
5102 \cs_new_protected:Npn \@@_actually_color:
5103 {
5104   \pgfpicture
5105   \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.

```

5106     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5107     {
5108         \pgfsetcornersarced
5109         {
5110             \pgfpoint
5111             { \l_@@_tab_rounded_corners_dim }
5112             { \l_@@_tab_rounded_corners_dim }
5113         }
5114     } % \end{macrocode}
5115 % Because we want \pkg{nicematrix} compatible with arrays constructed by
5116 % \pkg{array}, the nodes for the rows and columns (that is to say the nodes
5117 % |row-|\textsl{i} and |col-|\textsl{j}|) have not always the expected position,
5118 % that is to say, there is sometimes a slight shifting of something such as
5119 % |\arrayrulewidth|. Now, for the clipping, we have to change slightly the
5120 % position of that clipping whether a rounded rectangle around the array is
5121 % required. That's the point which is tested in the following line.
5122 % \begin{macrocode}
5123     \bool_if:NTF \l_@@_hvlines_bool
5124     {
5125         \pgfpathrectanglecorners
5126         {
5127             \pgfpointadd
5128             { \@@_qpoint:n { row-1 } }
5129             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5130         }
5131         {
5132             \pgfpointadd
5133             {
5134                 \@@_qpoint:n
5135                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5136             }
5137             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5138         }
5139     }
5140     {
5141         \pgfpathrectanglecorners
5142         { \@@_qpoint:n { row-1 } }
5143         {
5144             \pgfpointadd
5145             {
5146                 \@@_qpoint:n
5147                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5148             }
5149             { \pgfpoint \c_zero_dim \arrayrulewidth }
5150         }
5151     }
5152     \pgfusepath { clip }
5153 }
5154 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5155 {
5156     \begin { pgfscope }
5157         \@@_color_opacity ##2
5158         \use:c { g_@@_color _ ##1 _t1 }
5159         \tl_gclear:c { g_@@_color _ ##1 _t1 }
5160         \pgfusepath { fill }
5161     \end { pgfscope }
5162 }
5163 \endpgfpicture
5164 }
```

The following command will extract the potential key `opacity` in its optional argument (between

square brackets) and (of course) then apply the command `\color`.

```

5165 \cs_new_protected:Npn \@@_color_opacity
5166 {
5167   \peek_meaning:NTF [
5168     { \@@_color_opacity:w }
5169     { \@@_color_opacity:w [ ] }
5170   }

```

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

```

5171 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5172 {
5173   \tl_clear:N \l_tmpa_tl
5174   \keys_set_known:nn { 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.

```

5175   \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5176   \tl_if_empty:NTF \l_tmpb_tl
5177     { \@declaredcolor }
5178     { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5179 }

```

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

```

5180 \keys_define:nn { nicematrix / color-opacity }
5181 {
5182   opacity .tl_set:N          = \l_tmpa_tl ,
5183   opacity .value_required:n = true
5184 }

```

```

5185 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5186 {
5187   \tl_set:Nn \l_@@_rows_tl { #1 }
5188   \tl_set:Nn \l_@@_cols_tl { #2 }
5189   \@@_cartesian_path:
5190 }

```

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

```

5191 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5192 {
5193   \tl_if_blank:nF { #2 }
5194   {
5195     \@@_add_to_colors_seq:xn
5196     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5197     { \@@_cartesian_color:nn { #3 } { - } }
5198   }
5199 }

```

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

```

5200 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5201 {
5202   \tl_if_blank:nF { #2 }
5203   {
5204     \@@_add_to_colors_seq:xn
5205     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5206     { \@@_cartesian_color:nn { - } { #3 } }
5207   }
5208 }

```

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

```

5209 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5210 {
5211   \tl_if_blank:nF { #2 }
5212   {
5213     \@@_add_to_colors_seq:xn
5214     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5215     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5216   }
5217 }

```

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

```

5218 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5219 {
5220   \tl_if_blank:nF { #2 }
5221   {
5222     \@@_add_to_colors_seq:xn
5223     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5224     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5225   }
5226 }

```

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

```

5227 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5228 {
5229   \@@_cut_on_hyphen:w #1 \q_stop
5230   \tl_clear_new:N \l_@@_tmpc_tl
5231   \tl_clear_new:N \l_@@_tmpd_tl
5232   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5233   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5234   \@@_cut_on_hyphen:w #2 \q_stop
5235   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5236   \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`.

```

5237   \@@_cartesian_path:n { #3 }
5238 }

```

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

```

5239 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5240 {
5241   \clist_map_inline:nn { #3 }
5242   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5243 }

```

```

5244 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5245 {
5246   \int_step_inline:nn { \int_use:N \c@iRow }
5247   {
5248     \int_step_inline:nn { \int_use:N \c@jCol }
5249     {
5250       \int_if_even:nTF { ####1 + ##1 }
5251       { \@@_cellcolor [ #1 ] { #2 } }
5252       { \@@_cellcolor [ #1 ] { #3 } }
5253       { ##1 - ####1 }
5254     }
5255   }
5256 }

```

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”.

```

5257 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5258 {
5259   \@@_rectanglecolor [ #1 ] { #2 }
5260   { 1 - 1 }
5261   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5262 }

5263 \keys_define:nn { NiceMatrix / rowcolors }
5264 {
5265   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5266   respect-blocks .default:n = true ,
5267   cols .tl_set:N = \l_@@_cols_tl ,
5268   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5269   restart .default:n = true ,
5270   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5271 }

```

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]`.

In `nicematrix`, the command `\@@_rowcolors` apperas as a special case of `\@@_rowlistcolors`. `#1` (optional) is the color space ; `#2` is a list of intervals of rows ; `#3` is the list of colors ; `#4` is for the optional list of pairs *key=value*.

```

5272 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5273 {

```

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

```

5274   \group_begin:
5275   \seq_clear_new:N \l_@@_colors_seq
5276   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5277   \tl_clear_new:N \l_@@_cols_tl
5278   \tl_set:Nn \l_@@_cols_tl { - }
5279   \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).

```

5280   \int_zero_new:N \l_@@_color_int
5281   \int_set:Nn \l_@@_color_int 1
5282   \bool_if:NT \l_@@_respect_blocks_bool
5283   {

```

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`).

```

5284     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5285     \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5286     { \@@_not_in_exterior_p:nnnnn #1 }
5287   }
5288   \pgfpicture
5289   \pgf@relevantforpicturesizefalse

```

`#2` is the list of intervals of rows.

```

5290   \clist_map_inline:nn { #2 }
5291   {
5292     \tl_set:Nn \l_tmpa_tl { ##1 }
5293     \tl_if_in:NnTF \l_tmpa_tl { - }
5294     { \@@_cut_on_hyphen:w ##1 \q_stop }
5295     { \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.

```

5296 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5297 \int_set:Nn \l_@@_color_int
5298 { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5299 \int_zero_new:N \l_@@_tmpc_int
5300 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5301 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5302 {

```

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

```

5303 \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).

```

5304 \bool_if:NT \l_@@_respect_blocks_bool
5305 {
5306   \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5307   { \@@_intersect_our_row_p:nnnnn ###1 }
5308   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

5309 }
5310 \tl_set:Nx \l_@@_rows_tl
5311 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5312 \tl_clear_new:N \l_@@_color_tl
5313 \tl_set:Nx \l_@@_color_tl
5314 {
5315   \@@_color_index:n
5316   {
5317     \int_mod:nn
5318     { \l_@@_color_int - 1 }
5319     { \seq_count:N \l_@@_colors_seq }
5320     + 1
5321   }
5322 }
5323 \tl_if_empty:NF \l_@@_color_tl
5324 {
5325   \@@_add_to_colors_seq:xx
5326   { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5327   { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5328 }
5329 \int_incr:N \l_@@_color_int
5330 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5331 }
5332 }
5333 \endpgfpicture
5334 \group_end:
5335 }

```

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.

```

5336 \cs_new:Npn \@@_color_index:n #1
5337 {
5338   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5339   { \@@_color_index:n { #1 - 1 } }
5340   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5341 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5342 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5343 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

```

5344 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5345 {
5346   \int_compare:nNt { #3 } > \l_tmpb_int
5347   { \int_set:Nn \l_tmpb_int { #3 } }
5348 }

5349 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5350 {
5351   \bool_lazy_or:nnTF
5352   { \int_compare_p:nNn { #4 } = \c_zero_int }
5353   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5354   \prg_return_false:
5355   \prg_return_true:
5356 }

```

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

```

5357 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5358 {
5359   \bool_if:nTF
5360   {
5361     \int_compare_p:n { #1 <= \l_tmpa_int }
5362     &&
5363     \int_compare_p:n { \l_tmpa_int <= #3 }
5364   }
5365   \prg_return_true:
5366   \prg_return_false:
5367 }

```

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).

```

5368 \cs_new_protected:Npn \@@_cartesian_path:n #1
5369 {
5370   \bool_lazy_and:nnT
5371   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5372   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5373   {
5374     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5375     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5376   }

```

We begin the loop over the columns.

```

5377 \clist_map_inline:Nn \l_@@_cols_tl
5378 {
5379   \tl_set:Nn \l_tmpa_tl { ##1 }
5380   \tl_if_in:NnTF \l_tmpa_tl { - }
5381   { \@@_cut_on_hyphen:w ##1 \q_stop }
5382   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5383   \bool_lazy_or:nnT
5384   { \tl_if_blank_p:V \l_tmpa_tl }
5385   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5386   { \tl_set:Nn \l_tmpa_tl { 1 } }
5387   \bool_lazy_or:nnT
5388   { \tl_if_blank_p:V \l_tmpb_tl }
5389   { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5390   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5391   \int_compare:nNt \l_tmpb_tl > \c@jCol
5392   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

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

```
5393 \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.

```
5394 \@@_qpoint:n { col - \l_tmpa_tl }
5395 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5396 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5397 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5398 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5399 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
5400 \clist_map_inline:Nn \l_@@_rows_tl
5401 {
5402   \tl_set:Nn \l_tmpa_tl { #####1 }
5403   \tl_if_in:NnTF \l_tmpa_tl { - }
5404   { \@@_cut_on_hyphen:w #####1 \q_stop }
5405   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5406   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5407   \tl_if_empty:NT \l_tmpb_tl
5408   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5409   \int_compare:nNnT \l_tmpb_tl > \c@iRow
5410   { \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.

```
5411 \seq_if_in:NxF \l_@@_corners_cells_seq
5412 { \l_tmpa_tl - \l_@@_tmpc_tl }
5413 {
5414   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5415   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5416   \@@_qpoint:n { row - \l_tmpa_tl }
5417   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5418   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5419   \pgfpathrectanglecorners
5420   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5421   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5422 }
5423 }
5424 }
5425 }
```

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).

```
5426 \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.

```
5427 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5428 {
5429   \clist_set_eq:NN \l_tmpa_clist #1
5430   \clist_clear:N #1
5431   \clist_map_inline:Nn \l_tmpa_clist
5432   {
5433     \tl_set:Nn \l_tmpa_tl { ##1 }
5434     \tl_if_in:NnTF \l_tmpa_tl { - }
5435     { \@@_cut_on_hyphen:w ##1 \q_stop }
5436     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5437     \bool_lazy_or:nnT
5438     { \tl_if_blank_p:V \l_tmpa_tl }
5439     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
```



```

5440     { \tl_set:Nn \l_tmpa_tl { 1 } }
5441   \bool_lazy_or:nnT
5442     { \tl_if_blank_p:V \l_tmpb_tl }
5443     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5444     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5445   \int_compare:nNnT \l_tmpb_tl > #2
5446     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5447   \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5448     { \clist_put_right:Nn #1 { ###1 } }
5449 }
5450 }

```

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

```

5451 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5452 {
5453   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5454   {

```

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`).

```

5455     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5456     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5457   }
5458   \ignorespaces
5459 }

```

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

```

5460 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5461 {
5462   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5463   {
5464     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5465     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5466     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5467   }
5468   \ignorespaces
5469 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5470 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5471 { \@@_rowlistcolors_tabular [ #1 ] { #2 , #3 } }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5472 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5473 {
5474   \peek_remove_spaces:n
5475   {
5476     \tl_gput_right:Nx \g__nicematrix_pre_code_before_tl
5477     {
5478       \@@_rowlistcolors
5479       [ #1 ] { \int_use:N \c@iRow } { #2 } [ restart, #3 ]
5480     }
5481   }
5482 }

```

```

5483 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5484 {

```

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).

```
5485 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5486 {
```

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).

```
5487 \tl_gput_left:Nx \g_@@_pre_code_before_tl
5488 {
5489   \exp_not:N \columncolor [ #1 ]
5490   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5491 }
5492 }
5493 }
```

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`.

```
5494 \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`.

```
5495 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5496 {
5497   \int_compare:nNnTF \l_@@_first_col_int = 0
5498   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5499   {
5500     \int_compare:nNnTF \c@jCol = 0
5501     {
5502       \int_compare:nNnF \c@iRow = { -1 }
5503       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5504     }
5505     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5506   }
5507 }
```

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.

```
5508 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5509 {
5510   \int_compare:nNnF \c@iRow = 0
5511   {
5512     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5513     {
5514       \int_compare:nNnT \c@jCol > \c_zero_int
```

```

5515         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5516     }
5517 }
5518 }

```

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.

```

5519 \keys_define:nn { NiceMatrix / Rules }
5520 {
5521     position .int_set:N = \l_@@_position_int ,
5522     position .value_required:n = true ,
5523     start .int_set:N = \l_@@_start_int ,
5524     start .initial:n = 1 ,
5525     end .code:n =
5526         \bool_lazy_or:nnTF
5527         { \tl_if_empty_p:n { #1 } }
5528         { \str_if_eq_p:nn { #1 } { last } }
5529         { \int_set_eq:NN \l_@@_end_int \c@jCol }
5530         { \int_set:Nn \l_@@_end_int { #1 } }
5531 }

```

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.

```

5532 \keys_define:nn { NiceMatrix / RulesBis }
5533 {
5534     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5535     multiplicity .initial:n = 1 ,
5536     dotted .bool_set:N = \l_@@_dotted_bool ,
5537     dotted .initial:n = false ,
5538     dotted .default:n = true ,
5539     color .code:n = \@@_set_CT@arc@:n { #1 } ,
5540     color .value_required:n = true ,
5541     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5542     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`.

```

5543     tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5544     tikz .value_required:n = true ,
5545     tikz .initial:n = ,
5546     total-width .dim_set:N = \l_@@_rule_width_dim ,
5547     total-width .value_required:n = true ,
5548     width .meta:n = { total-width = #1 } ,
5549     unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5550 }

```

The vertical rules

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

```

5551 \cs_new_protected:Npn \@@_vline:n #1
5552 {

```

The group is for the options.

```

5553 \group_begin:
5554 \int_zero_new:N \l_@@_end_int
5555 \int_set_eq:NN \l_@@_end_int \c@iRow
5556 \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).

```

5557 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5558 \@@_vline_i:
5559 \group_end:
5560 }

5561 \cs_new_protected:Npn \@@_vline_i:
5562 {
5563 \int_zero_new:N \l_@@_local_start_int
5564 \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`.

```

5565 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5566 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5567 \l_tmpa_tl
5568 {

```

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.

```

5569 \bool_gset_true:N \g_tmpa_bool
5570 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5571 { \@@_test_vline_in_block:nnnnn #1 }
5572 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5573 { \@@_test_vline_in_block:nnnnn #1 }
5574 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5575 { \@@_test_vline_in_stroken_block:nnnn #1 }
5576 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5577 \bool_if:NTF \g_tmpa_bool
5578 {
5579 \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.

```

5580 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5581 }
5582 {
5583 \int_compare:nNnT \l_@@_local_start_int > 0
5584 {
5585 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5586 \@@_vline_ii:
5587 \int_zero:N \l_@@_local_start_int
5588 }
5589 }
5590 }
5591 \int_compare:nNnT \l_@@_local_start_int > 0
5592 {
5593 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5594 \@@_vline_ii:
5595 }
5596 }

```

```

5597 \cs_new_protected:Npn \@@_test_in_corner_v:
5598 {
5599 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }

```

```

5600 {
5601   \seq_if_in:NxT
5602     \l_@@_corners_cells_seq
5603     { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5604     { \bool_set_false:N \g_tmpa_bool }
5605 }
5606 {
5607   \seq_if_in:NxT
5608     \l_@@_corners_cells_seq
5609     { \l_tmpa_tl - \l_tmpb_tl }
5610     {
5611       \int_compare:nNnTF \l_tmpb_tl = 1
5612       { \bool_set_false:N \g_tmpa_bool }
5613       {
5614         \seq_if_in:NxT
5615           \l_@@_corners_cells_seq
5616           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5617           { \bool_set_false:N \g_tmpa_bool }
5618       }
5619     }
5620 }
5621 }

5622 \cs_new_protected:Npn \@@_vline_ii:
5623 {
5624   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5625   \bool_if:NTF \l_@@_dotted_bool
5626     \@@_vline_iv:
5627     {
5628       \tl_if_empty:NTF \l_@@_tikz_rule_tl
5629       \@@_vline_iii:
5630       \@@_vline_v:
5631     }
5632 }

```

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

```

5633 \cs_new_protected:Npn \@@_vline_iii:
5634 {
5635   \pgfpicture
5636   \pgfrememberpicturepositiononpagetrue
5637   \pgf@relevantforpicturesizefalse
5638   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5639   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5640   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5641   \dim_set:Nn \l_tmpb_dim
5642     {
5643       \pgf@x
5644       - 0.5 \l_@@_rule_width_dim
5645       +
5646       ( \arrayrulewidth * \l_@@_multiplicity_int
5647         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5648     }
5649   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5650   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5651   \bool_lazy_all:nT
5652     {
5653       { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5654       { \cs_if_exist_p:N \CT@drsc@ }
5655       { ! \tl_if_blank_p:V \CT@drsc@ }
5656     }
5657     {
5658       \group_begin:

```

```

5659 \CT@drsc@
5660 \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5661 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5662 \dim_set:Nn \l_@@_tmpd_dim
5663 {
5664     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5665     * ( \l_@@_multiplicity_int - 1 )
5666 }
5667 \pgfpathrectanglecorners
5668 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5669 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5670 \pgfusepath { fill }
5671 \group_end:
5672 }
5673 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5674 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5675 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5676 {
5677     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5678     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5679     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5680     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5681 }
5682 \CT@arc@
5683 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5684 \pgfsetrectcap
5685 \pgfusepathqstroke
5686 \endpgfpicture
5687 }

```

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

```

5688 \cs_new_protected:Npn \@@_vline_iv:
5689 {
5690     \pgfpicture
5691     \pgfrememberpicturepositiononpagetrue
5692     \pgf@relevantforpicturesizefalse
5693     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5694     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5695     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5696     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5697     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5698     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5699     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5700     \CT@arc@
5701     \@@_draw_line:
5702     \endpgfpicture
5703 }

```

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`).

```

5704 \cs_new_protected:Npn \@@_vline_v:
5705 {
5706     \begin {tikzpicture }
5707     \pgfrememberpicturepositiononpagetrue
5708     \pgf@relevantforpicturesizefalse
5709     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5710     \dim_set_eq:NN \l_tmpa_dim \pgf@y
5711     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5712     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5713     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5714     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5715     \exp_args:NV \tikzset \l_@@_tikz_rule_tl

```

```

5716 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5717 ( \l_tmpb_dim , \l_tmpa_dim ) --
5718 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5719 \end { tikzpicture }
5720 }

```

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).

```

5721 \cs_new_protected:Npn \@@_draw_vlines:
5722 {
5723   \int_step_inline:nnn
5724   {
5725     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5726     1 2
5727   }
5728   {
5729     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5730     { \int_eval:n { \c@jCol + 1 } }
5731     \c@jCol
5732   }
5733   {
5734     \tl_if_eq:NnF \l_@@_vlines_clist { all }
5735     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5736     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5737   }
5738 }

```

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}`.

```

5739 \cs_new_protected:Npn \@@_hline:n #1
5740 {

```

The group is for the options.

```

5741   \group_begin:
5742   \int_zero_new:N \l_@@_end_int
5743   \int_set_eq:NN \l_@@_end_int \c@jCol
5744   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5745   \@@_hline_i:
5746   \group_end:
5747 }
5748 \cs_new_protected:Npn \@@_hline_i:
5749 {
5750   \int_zero_new:N \l_@@_local_start_int
5751   \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`.

```

5752   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5753   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5754   \l_tmpb_tl
5755   {

```

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.

```

5756   \bool_gset_true:N \g_tmpa_bool
5757   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5758   { \@@_test_hline_in_block:nnnnn ##1 }
5759   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq

```

```

5760     { \@@_test_hline_in_block:nnnnn ##1 }
5761     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5762     { \@@_test_hline_in_stroken_block:nnnn ##1 }
5763     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5764     \bool_if:NTF \g_tmpa_bool
5765     {
5766         \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.

```

5767         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5768     }
5769     {
5770         \int_compare:nNnT \l_@@_local_start_int > 0
5771         {
5772             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5773             \@@_hline_ii:
5774             \int_zero:N \l_@@_local_start_int
5775         }
5776     }
5777 }
5778 \int_compare:nNnT \l_@@_local_start_int > 0
5779 {
5780     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5781     \@@_hline_ii:
5782 }
5783 }

```

```

5784 \cs_new_protected:Npn \@@_test_in_corner_h:
5785 {
5786     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5787     {
5788         \seq_if_in:NxT
5789         \l_@@_corners_cells_seq
5790         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5791         { \bool_set_false:N \g_tmpa_bool }
5792     }
5793     {
5794         \seq_if_in:NxT
5795         \l_@@_corners_cells_seq
5796         { \l_tmpa_tl - \l_tmpb_tl }
5797         {
5798             \int_compare:nNnTF \l_tmpa_tl = 1
5799             { \bool_set_false:N \g_tmpa_bool }
5800             {
5801                 \seq_if_in:NxT
5802                 \l_@@_corners_cells_seq
5803                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5804                 { \bool_set_false:N \g_tmpa_bool }
5805             }
5806         }
5807     }
5808 }

```

```

5809 \cs_new_protected:Npn \@@_hline_ii:
5810 {
5811     % \bool_set_false:N \l_@@_dotted_bool
5812     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5813     \bool_if:NTF \l_@@_dotted_bool
5814     \@@_hline_iv:
5815     {
5816         \tl_if_empty:NTF \l_@@_tikz_rule_tl

```



```

5817         \@@_hline_iii:
5818         \@@_hline_v:
5819     }
5820 }

```

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

```

5821 \cs_new_protected:Npn \@@_hline_iii:
5822 {
5823     \pgfpicture
5824     \pgfrememberpicturepositiononpagetrue
5825     \pgf@relevantforpicturesizefalse
5826     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5827     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5828     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5829     \dim_set:Nn \l_tmpb_dim
5830     {
5831         \pgf@y
5832         - 0.5 \l_@@_rule_width_dim
5833         +
5834         ( \arrayrulewidth * \l_@@_multiplicity_int
5835           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5836     }
5837     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5838     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5839     \bool_lazy_all:nT
5840     {
5841         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5842         { \cs_if_exist_p:N \CT@drsc@ }
5843         { ! \tl_if_blank_p:V \CT@drsc@ }
5844     }
5845     {
5846         \group_begin:
5847         \CT@drsc@
5848         \dim_set:Nn \l_@@_tmpd_dim
5849         {
5850             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5851             * ( \l_@@_multiplicity_int - 1 )
5852         }
5853         \pgfpathrectanglecorners
5854         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5855         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5856         \pgfusepathqfill
5857         \group_end:
5858     }
5859     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5860     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5861     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5862     {
5863         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5864         \dim_sub:Nn \l_tmpb_dim \doublerulesep
5865         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5866         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5867     }
5868     \CT@arc@
5869     \pgfsetlinewidth { 1.1 \arrayrulewidth }
5870     \pgfsetrectcap
5871     \pgfusepathqstroke
5872     \endpgfpicture
5873 }

```

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}

```

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

```

5874 \cs_new_protected:Npn \@@_hline_iv:
5875 {
5876   \pgfpicture
5877   \pgfrememberpicturepositiononpagetrue
5878   \pgf@relevantforpicturesizefalse
5879   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5880   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5881   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
5882   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5883   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5884   \int_compare:nNnT \l_@@_local_start_int = 1
5885   {
5886     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5887     \bool_if:NF \g_@@_delims_bool
5888     { \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.

```

5889   \tl_if_eq:NnF \g_@@_left_delim_tl (
5890     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
5891   )
5892   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5893   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5894   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5895   {
5896     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5897     \bool_if:NF \g_@@_delims_bool
5898     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5899     \tl_if_eq:NnF \g_@@_right_delim_tl )
5900     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5901   }
5902   \CT@arc@
5903   \@@_draw_line:
5904   \endpgfpicture
5905 }

```

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`).

```

5906 \cs_new_protected:Npn \@@_hline_v:
5907 {
5908   \begin { tikzpicture }
5909   \pgfrememberpicturepositiononpagetrue
5910   \pgf@relevantforpicturesizefalse
5911   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5912   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5913   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5914   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }

```

```

5915 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5916 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5917 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5918 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5919 ( \l_tmpa_dim , \l_tmpb_dim ) --
5920 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5921 \end { tikzpicture }
5922 }

```

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)).

```

5923 \cs_new_protected:Npn \@@_draw_hlines:
5924 {
5925   \int_step_inline:nnn
5926   {
5927     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5928     1 2
5929   }
5930   {
5931     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5932     { \int_eval:n { \c@iRow + 1 } }
5933     \c@iRow
5934   }
5935   {
5936     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5937     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5938     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5939   }
5940 }

```

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

```

5941 \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.

```

5942 \cs_set:Npn \@@_Hline_i:n #1
5943 {
5944   \peek_remove_spaces:n
5945   {
5946     \peek_meaning:NTF \Hline
5947     { \@@_Hline_ii:nn { #1 + 1 } }
5948     { \@@_Hline_iii:n { #1 } }
5949   }
5950 }
5951 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5952 \cs_set:Npn \@@_Hline_iii:n #1
5953 {
5954   \peek_meaning:NTF [
5955   { \@@_Hline_iv:nw { #1 } }
5956   { \@@_Hline_iv:nw { #1 } [ ] }
5957 }
5958 \cs_set:Npn \@@_Hline_iv:nw #1 [ #2 ]
5959 {
5960   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
5961   \skip_vertical:n { \l_@@_rule_width_dim }
5962   \tl_gput_right:Nx \g_@@_pre_code_after_tl
5963   {
5964     \@@_hline:n
5965     {
5966       multiplicity = #1 ,
5967       position = \int_eval:n { \c@iRow + 1 } ,

```

```

5968         total-width = \dim_use:N \l_@@_rule_width_dim ,
5969         #2
5970     }
5971 }
5972 \egroup
5973 }

```

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).

```

5974 \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`).

```

5975 \cs_new_protected:Npn \@@_custom_line:n #1
5976 {
5977     \str_clear_new:N \l_@@_command_str
5978     \str_clear_new:N \l_@@_ccommand_str
5979     \str_clear_new:N \l_@@_letter_str
5980     \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.

```

5981     \bool_lazy_all:nTF
5982     {
5983         { \str_if_empty_p:N \l_@@_letter_str }
5984         { \str_if_empty_p:N \l_@@_command_str }
5985         { \str_if_empty_p:N \l_@@_ccommand_str }
5986     }
5987     { \@@_error:n { No~letter~and~no~command } }
5988     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
5989 }

5990 \keys_define:nn { NiceMatrix / custom-line }
5991 {
5992     letter .str_set:N = \l_@@_letter_str ,
5993     letter .value_required:n = true ,
5994     command .str_set:N = \l_@@_command_str ,
5995     command .value_required:n = true ,
5996     ccommand .str_set:N = \l_@@_ccommand_str ,
5997     ccommand .value_required:n = true ,
5998 }

```

```

5999 \cs_new_protected:Npn \@@_custom_line_i:n #1
6000 {

```

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

```

6001     \bool_set_false:N \l_@@_tikz_rule_bool
6002     \bool_set_false:N \l_@@_dotted_rule_bool
6003     \bool_set_false:N \l_@@_color_bool

```

```

6004 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6005 \bool_if:NT \l_@@_tikz_rule_bool
6006 {
6007   \IfPackageLoadedTF { tikz }
6008   { }
6009   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6010   \bool_if:NT \l_@@_color_bool
6011   { \@@_error:n { color~in~custom~line~with~tikz } }
6012 }
6013 \bool_if:nT
6014 {
6015   \int_compare_p:nNn \l_@@_multiplicity_int > 1
6016   && \l_@@_dotted_rule_bool
6017 }
6018 { \@@_error:n { key~multiplicity~with~dotted } }
6019 \str_if_empty:NF \l_@@_letter_str
6020 {
6021   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6022   { \@@_error:n { Several~letters } }
6023   {
6024     \exp_args:NnV \tl_if_in:NnTF
6025     \c_@@_forbidden_letters_str \l_@@_letter_str
6026     { \@@_error:n { Forbidden~letter } }
6027     {

```

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.

```

6028 \keys_define:nx { NiceMatrix / ColumnTypes }
6029 {
6030   \l_@@_letter_str .code:n =
6031   { \@@_v_custom_line:n { \exp_not:n { #1 } } }
6032 }
6033 }
6034 }
6035 }
6036 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6037 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6038 }
6039 \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.

```

6040 \keys_define:nn { NiceMatrix / custom-line-bis }
6041 {
6042   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6043   multiplicity .initial:n = 1 ,
6044   multiplicity .value_required:n = true ,
6045   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6046   color .value_required:n = true ,
6047   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6048   tikz .value_required:n = true ,
6049   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6050   dotted .value_forbidden:n = true ,
6051   total-width .code:n = { } ,
6052   total-width .value_required:n = true ,
6053   width .code:n = { } ,
6054   width .value_required:n = true ,
6055   sep-color .code:n = { } ,
6056   sep-color .value_required:n = true ,
6057   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6058 }

```

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

```
6059 \bool_new:N \l_@@_dotted_rule_bool
6060 \bool_new:N \l_@@_tikz_rule_bool
6061 \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`.

```
6062 \keys_define:nn { NiceMatrix / custom-line-width }
6063 {
6064   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6065   multiplicity .initial:n = 1 ,
6066   multiplicity .value_required:n = true ,
6067   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6068   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6069                       \bool_set_true:N \l_@@_total_width_bool ,
6070   total-width .value_required:n = true ,
6071   width .meta:n = { total-width = #1 } ,
6072   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6073 }
```

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`).

```
6074 \cs_new_protected:Npn \@@_h_custom_line:n #1
6075 {
```

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`.

```
6076   \cs_set:cpn { nicematrix - \l_@@_command_str }
6077   {
6078     \noalign
6079     {
6080       \@@_compute_rule_width:n { #1 }
6081       \skip_vertical:n { \l_@@_rule_width_dim }
6082       \tl_gput_right:Nx \g_@@_pre_code_after_tl
6083       {
6084         \@@_hline:n
6085         {
6086           #1 ,
6087           position = \int_eval:n { \c@iRow + 1 } ,
6088           total-width = \dim_use:N \l_@@_rule_width_dim
6089         }
6090       }
6091     }
6092   }
6093   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6094 }
6095 \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`).

```
6096 \cs_new_protected:Npn \@@_c_custom_line:n #1
6097 {
```

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

```
6098   \exp_args:Nc \NewExpandableDocumentCommand
6099   { nicematrix - \l_@@_ccommand_str }
6100   { 0 { } m }
```

```

6101 {
6102   \noalign
6103   {
6104     \@@_compute_rule_width:n { #1 , ##1 }
6105     \skip_vertical:n { \l_@@_rule_width_dim }
6106     \clist_map_inline:nn
6107       { ##2 }
6108       { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6109   }
6110 }
6111 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6112 }

```

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*.

```

6113 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6114 {
6115   \str_if_in:nnTF { #2 } { - }
6116   { \@@_cut_on_hyphen:w #2 \q_stop }
6117   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6118   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6119   {
6120     \@@_hline:n
6121     {
6122       #1 ,
6123       start = \l_tmpa_tl ,
6124       end = \l_tmpb_tl ,
6125       position = \int_eval:n { \c@iRow + 1 } ,
6126       total-width = \dim_use:N \l_@@_rule_width_dim
6127     }
6128   }
6129 }
6130 \cs_generate_variant:Nn \@@_c_custom_line:nn { n V }
6131 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6132 {
6133   \bool_set_false:N \l_@@_tikz_rule_bool
6134   \bool_set_false:N \l_@@_total_width_bool
6135   \bool_set_false:N \l_@@_dotted_rule_bool
6136   \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6137   \bool_if:NF \l_@@_total_width_bool
6138   {
6139     \bool_if:NTF \l_@@_dotted_rule_bool
6140     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6141     {
6142       \bool_if:NF \l_@@_tikz_rule_bool
6143       {
6144         \dim_set:Nn \l_@@_rule_width_dim
6145         {
6146           \arrayrulewidth * \l_@@_multiplicity_int
6147           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6148         }
6149       }
6150     }
6151   }
6152 }
6153 \cs_new_protected:Npn \@@_v_custom_line:n #1
6154 {
6155   \@@_compute_rule_width:n { #1 }

```

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

```

6156   \tl_gput_right:Nx \g_@@_preamble_tl
6157   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6158   \tl_gput_right:Nx \g_@@_pre_code_after_tl

```

```

6159     {
6160         \@@_vline:n
6161         {
6162             #1 ,
6163             position = \int_eval:n { \c@jCol + 1 } ,
6164             total-width = \dim_use:N \l_@@_rule_width_dim
6165         }
6166     }
6167 }
6168 \@@_custom_line:n
6169 { 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.

```

6170 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6171 {
6172     \bool_lazy_all:nT
6173     {
6174         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6175         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6176         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6177         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6178     }
6179     { \bool_gset_false:N \g_tmpa_bool }
6180 }

```

The same for vertical rules.

```

6181 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6182 {
6183     \bool_lazy_all:nT
6184     {
6185         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6186         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6187         { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6188         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6189     }
6190     { \bool_gset_false:N \g_tmpa_bool }
6191 }
6192 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6193 {
6194     \bool_lazy_all:nT
6195     {
6196         {
6197             ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6198             || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6199         }
6200         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6201         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6202     }
6203     { \bool_gset_false:N \g_tmpa_bool }
6204 }
6205 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6206 {
6207     \bool_lazy_all:nT
6208     {
6209         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6210         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6211         {

```



```

6212         ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6213         || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6214     }
6215 }
6216 { \bool_gset_false:N \g_tmpa_bool }
6217 }

```

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.

```

6218 \cs_new_protected:Npn \@@_compute_corners:
6219 {

```

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.

```

6220     \seq_clear_new:N \l_@@_corners_cells_seq
6221     \clist_map_inline:Nn \l_@@_corners_clist
6222     {
6223         \str_case:nnF { ##1 }
6224         {
6225             { NW }
6226             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6227             { NE }
6228             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6229             { SW }
6230             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6231             { SE }
6232             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6233         }
6234         { \@@_error:nn { bad~corner } { ##1 } }
6235     }

```

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

```

6236     \seq_if_empty:NF \l_@@_corners_cells_seq
6237     {

```

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.

```

6238         \tl_gput_right:Nx \g_@@_aux_tl
6239         {
6240             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6241             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6242         }
6243     }
6244 }

```

“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.

```

6245 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6246 {

```

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.

```

6247 \bool_set_false:N \l_tmpa_bool
6248 \int_zero_new:N \l_@@_last_empty_row_int
6249 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6250 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6251 {
6252   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6253   \bool_lazy_or:nnTF
6254   {
6255     \cs_if_exist_p:c
6256     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6257   }
6258   \l_tmpb_bool
6259   { \bool_set_true:N \l_tmpa_bool }
6260   {
6261     \bool_if:NF \l_tmpa_bool
6262     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6263   }
6264 }

```

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

```

6265 \bool_set_false:N \l_tmpa_bool
6266 \int_zero_new:N \l_@@_last_empty_column_int
6267 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6268 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6269 {
6270   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6271   \bool_lazy_or:nnTF
6272   \l_tmpb_bool
6273   {
6274     \cs_if_exist_p:c
6275     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6276   }
6277   { \bool_set_true:N \l_tmpa_bool }
6278   {
6279     \bool_if:NF \l_tmpa_bool
6280     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6281   }
6282 }

```

Now, we loop over the rows.

```

6283 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6284 {

```

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

```

6285 \bool_set_false:N \l_tmpa_bool
6286 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6287 {
6288   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6289   \bool_lazy_or:nnTF
6290   \l_tmpb_bool
6291   {
6292     \cs_if_exist_p:c
6293     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6294   }
6295   { \bool_set_true:N \l_tmpa_bool }
6296   {
6297     \bool_if:NF \l_tmpa_bool

```

```

6298         {
6299             \int_set:Nn \l_@@_last_empty_column_int { ####1 }
6300             \seq_put_right:Nn
6301                 \l_@@_corners_cells_seq
6302                 { ##1 - ####1 }
6303         }
6304     }
6305 }
6306 }
6307 }

```

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`).

```

6308 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6309 {
6310     \int_set:Nn \l_tmpa_int { #1 }
6311     \int_set:Nn \l_tmpb_int { #2 }
6312     \bool_set_false:N \l_tmpb_bool
6313     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6314         { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6315 }
6316 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6317 {
6318     \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6319     {
6320         \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6321         {
6322             \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6323             {
6324                 \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6325                 { \bool_set_true:N \l_tmpb_bool }
6326             }
6327         }
6328     }
6329 }

```

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.

```

6330 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6331 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6332 {
6333     auto-columns-width .code:n =
6334     {
6335         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6336         \dim_gzero_new:N \g_@@_max_cell_width_dim
6337         \bool_set_true:N \l_@@_auto_columns_width_bool
6338     }
6339 }

```

```

6340 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6341 {
6342   \int_gincr:N \g_@@_NiceMatrixBlock_int
6343   \dim_zero:N \l_@@_columns_width_dim
6344   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6345   \bool_if:NT \l_@@_block_auto_columns_width_bool
6346   {
6347     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6348     {
6349       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
6350       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6351     }
6352   }
6353 }

```

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).

```

6354 {
6355   \bool_if:NT \l_@@_block_auto_columns_width_bool
6356   {
6357     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6358     \iow_shipout:Nx \@mainaux
6359     {
6360       \cs_gset:cpn
6361       { @@_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.

```

6362       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6363     }
6364     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6365   }
6366 }

```

25 The extra nodes

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

```

6367 \cs_generate_variant:Nn \dim_min:nn { v n }
6368 \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).

```

6369 \cs_new_protected:Npn \@@_create_extra_nodes:
6370 {
6371   \bool_if:nTF \l_@@_medium_nodes_bool
6372   {
6373     \bool_if:nTF \l_@@_large_nodes_bool
6374     \@@_create_medium_and_large_nodes:
6375     \@@_create_medium_nodes:
6376   }
6377   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6378 }

```

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`.

```

6379 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6380 {
6381   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6382   {
6383     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6384     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6385     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6386     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6387   }
6388   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6389   {
6390     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6391     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6392     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6393     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6394   }

```

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

```

6395   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6396   {
6397     \int_step_variable:nnNn
6398     \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.

```

6399     {
6400       \cs_if_exist:cT
6401       { 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`.

```

6402     {
6403       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6404       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6405       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6406       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6407       {
6408         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6409         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6410       }

```

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`.

```

6411       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6412       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6413       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6414       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6415       {
6416         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }

```

```

6417         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6418     }
6419 }
6420 }
6421 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6422 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6423 {
6424     \dim_compare:nNnT
6425     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6426     {
6427         \@@_qpoint:n { row - \@@_i: - base }
6428         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6429         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6430     }
6431 }
6432 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6433 {
6434     \dim_compare:nNnT
6435     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6436     {
6437         \@@_qpoint:n { col - \@@_j: }
6438         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6439         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6440     }
6441 }
6442 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6443 \cs_new_protected:Npn \@@_create_medium_nodes:
6444 {
6445     \pgfpicture
6446     \pgfrememberpicturepositiononpagetrue
6447     \pgf@relevantforpicturesizefalse
6448     \@@_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”.

```

6449     \tl_set:Nn \l_@@_suffix_tl { -medium }
6450     \@@_create_nodes:
6451     \endpgfpicture
6452 }

```

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:`.

```

6453 \cs_new_protected:Npn \@@_create_large_nodes:
6454 {
6455     \pgfpicture
6456     \pgfrememberpicturepositiononpagetrue
6457     \pgf@relevantforpicturesizefalse
6458     \@@_computations_for_medium_nodes:
6459     \@@_computations_for_large_nodes:
6460     \tl_set:Nn \l_@@_suffix_tl { - large }
6461     \@@_create_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6462 \endpgfpicture
6463 }
6464 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6465 {
6466   \pgfpicture
6467   \pgfrememberpicturepositiononpagetrue
6468   \pgf@relevantforpicturesizefalse
6469   \@@_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”.

```

6470   \tl_set:Nn \l_@@_suffix_tl { - medium }
6471   \@@_create_nodes:
6472   \@@_computations_for_large_nodes:
6473   \tl_set:Nn \l_@@_suffix_tl { - large }
6474   \@@_create_nodes:
6475 \endpgfpicture
6476 }

```

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.

```

6477 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6478 {
6479   \int_set:Nn \l_@@_first_row_int 1
6480   \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`.

```

6481   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6482   {
6483     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6484     {
6485       (
6486         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6487         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6488       )
6489       / 2
6490     }
6491     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6492     { l_@@_row _ \@@_i: _ min _ dim }
6493   }
6494   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6495   {
6496     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6497     {
6498       (
6499         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6500         \dim_use:c
6501         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6502       )
6503       / 2
6504     }
6505     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6506     { l_@@_column _ \@@_j: _ max _ dim }
6507   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6508   \dim_sub:cn
6509   { l_@@_column _ 1 _ min _ dim }
6510   \l_@@_left_margin_dim
6511   \dim_add:cn
6512   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6513   \l_@@_right_margin_dim
6514 }

```

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`).

```

6515 \cs_new_protected:Npn \@@_create_nodes:
6516 {
6517   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6518   {
6519     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6520     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6521       \@@_pgf_rect_node:nnnnn
6522       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6523       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6524       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6525       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6526       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6527       \str_if_empty:NF \l_@@_name_str
6528       {
6529         \pgfnodealias
6530         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6531         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6532       }
6533     }
6534   }

```

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 .

```

6535   \cs_if_exist_use:NF
6536   \seq_map_pairwise_function:NNN
6537   \seq_mapthread_function:NNN
6538   \g_@@_multicolumn_cells_seq
6539   \g_@@_multicolumn_sizes_seq
6540   \@@_node_for_multicolumn:nn
6541 }

```

```

6542 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6543 {
6544   \cs_set_nopar:Npn \@@_i: { #1 }
6545   \cs_set_nopar:Npn \@@_j: { #2 }
6546 }

```

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”).

```

6547 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6548 {
6549   \@@_extract_coords_values: #1 \q_stop
6550   \@@_pgf_rect_node:nnnnn
6551   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6552   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6553   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6554   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
6555   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6556   \str_if_empty:NF \l_@@_name_str
6557   {
6558     \pgfnodealias
6559     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6560     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}

```



```

6561     }
6562 }

```

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.

```

6563 \keys_define:nn { NiceMatrix / Block / FirstPass }
6564 {
6565   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6566   l .value_forbidden:n = true ,
6567   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6568   r .value_forbidden:n = true ,
6569   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6570   c .value_forbidden:n = true ,
6571   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6572   L .value_forbidden:n = true ,
6573   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6574   R .value_forbidden:n = true ,
6575   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6576   C .value_forbidden:n = true ,
6577   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6578   t .value_forbidden:n = true ,
6579   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6580   b .value_forbidden:n = true ,
6581   color .tl_set:N = \l_@@_color_tl ,
6582   color .value_required:n = true ,
6583   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6584   respect-arraystretch .default:n = true ,
6585 }

```

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.

```

6586 \cs_new_protected:Npn \@@_block:n #1
6587 {
6588   \peek_meaning:NTF [
6589     { \@@_block_i:n { #1 } }
6590     { \@@_Block_i:[#1] }
6591   ]
6592
6593   \cs_new_protected:Npn \@@_block_i:n #1[#2] { \@@_block:n {#1,#2} }
6594
6595   \cs_new_protected:Npn \@@_Block: { \@@_block:n { } }

```



```

6596 \NewExpandableDocumentCommand \@@_Block_i: { 0 { } m D < > { } +m }
6597 {

```

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).

```

6598   \peek_remove_spaces:n
6599   {
6600     \tl_if_blank:nTF { #2 }
6601     { \@@_Block_i 1-1 \q_stop }
6602     {

```

```

6603         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6604         \@@_Block_i_czech \@@_Block_i
6605         #2 \q_stop
6606     }
6607     { #1 } { #3 } { #4 }
6608 }
6609 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6610 \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`.

```

6611 {
6612     \char_set_catcode_active:N -
6613     \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6614 }

```

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.

```

6615 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6616 {

```

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).

```

6617     \bool_lazy_or:nnTF
6618     { \tl_if_blank_p:n { #1 } }
6619     { \str_if_eq_p:nn { #1 } { * } }
6620     { \int_set:Nn \l_tmpa_int { 100 } }
6621     { \int_set:Nn \l_tmpa_int { #1 } }
6622     \bool_lazy_or:nnTF
6623     { \tl_if_blank_p:n { #2 } }
6624     { \str_if_eq_p:nn { #2 } { * } }
6625     { \int_set:Nn \l_tmpb_int { 100 } }
6626     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6627     \int_compare:nNnTF \l_tmpb_int = 1
6628     {
6629         \str_if_empty:Ntf \l_@@_hpos_cell_str
6630         { \str_set:Nn \l_@@_hpos_block_str c }
6631         { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6632     }
6633     { \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.

```

6634     \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6635     \tl_set:Nx \l_tmpa_tl
6636     {
6637         { \int_use:N \c@iRow }
6638         { \int_use:N \c@jCol }
6639         { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6640         { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6641     }

```

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 curryfication).

```

6642 \bool_if:nTF
6643 {
6644   (
6645     \int_compare_p:nNn { \l_tmpa_int } = 1
6646     ||
6647     \int_compare_p:nNn { \l_tmpb_int } = 1
6648   )
6649   && ! \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.

```

6650   && ! \l_@@_X_column_bool
6651 }
6652 { \exp_args:Nxx \@@_Block_iv:nnnnn }
6653 { \exp_args:Nxx \@@_Block_v:nnnnn }
6654 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6655 }

```

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).

```

6656 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6657 {
6658   \int_gincr:N \g_@@_block_box_int
6659   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6660   {
6661     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6662     {
6663       \@@_actually_diagbox:nnnnnn
6664       { \int_use:N \c@iRow }
6665       { \int_use:N \c@jCol }
6666       { \int_eval:n { \c@iRow + #1 - 1 } }
6667       { \int_eval:n { \c@jCol + #2 - 1 } }
6668       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6669     }
6670   }
6671   \box_gclear_new:c
6672   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
6673   \hbox_gset:cn
6674   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
6675   {

```

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}`).

```

6676   \tl_if_empty:NTF \l_@@_color_tl
6677   { \int_compare:nNnT { #2 } = 1 \set@color }
6678   { \@@_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`.

```

6679     \int_compare:nNnT { #1 } = 1
6680     {
6681         \int_compare:nNnTF \c@iRow = 0
6682         \l_@@_code_for_first_row_tl
6683         {
6684             \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6685             \l_@@_code_for_last_row_tl
6686         }
6687         \g_@@_row_style_tl
6688     }
6689 \group_begin:
6690 \bool_if:NF \l_@@_respect_arraystretch_bool
6691 { \cs_set:Npn \arraystretch { 1 } }
6692 \dim_zero:N \extrarowheight
6693 #4

```

If the box is rotated (the key `\rotate` may be in the previous #4), the tabular used for the content of the cell will be constructed with a format `c`. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

```

6694 \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
6695 \bool_if:NTF \l_@@_tabular_bool
6696 {
6697     \bool_lazy_all:nTF
6698     {
6699         { \int_compare_p:nNn { #2 } = 1 }
6700         { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6701         { ! \g_@@_rotate_bool }
6702     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

6703     {
6704         \use:x
6705         {
6706             \exp_not:N \begin { minipage }%
6707             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6708             { \l_@@_col_width_dim }
6709             \str_case:Vn \l_@@_hpos_block_str
6710             {
6711                 c \centering
6712                 r \raggedleft
6713                 l \raggedright
6714             }
6715         }
6716         #5
6717         \end { minipage }
6718     }
6719     {
6720         \use:x
6721         {
6722             \exp_not:N \begin { tabular }%
6723             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6724             { @ { } \l_@@_hpos_block_str @ { } }
6725         }
6726         #5
6727         \end { tabular }
6728     }
6729 }
6730 {
6731     \c_math_toggle_token

```

```

6732         \use:x
6733         {
6734             \exp_not:N \begin { array }%
6735             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6736             { @ { } \l_@@_hpos_block_str @ { } }
6737         }
6738         #5
6739         \end { array }
6740         \c_math_toggle_token
6741     }
6742     \group_end:
6743 }
6744 \bool_if:NT \g_@@_rotate_bool
6745 {
6746     \box_grotate:cn
6747     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6748     { 90 }
6749     \bool_if:NT \g_@@_rotate_c_bool
6750     {
6751         \hbox_gset:cn
6752         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6753         {
6754             \c_math_toggle_token
6755             \vcenter
6756             {
6757                 \box_use:c
6758                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6759             }
6760             \c_math_toggle_token
6761         }
6762     }
6763     \bool_gset_false:N \g_@@_rotate_bool
6764 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

6765     \int_compare:nNnT { #2 } = 1
6766     {
6767         \dim_gset:Nn \g_@@_blocks_wd_dim
6768         {
6769             \dim_max:nn
6770             \g_@@_blocks_wd_dim
6771             {
6772                 \box_wd:c
6773                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6774             }
6775         }
6776     }

```

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.

```

6777     \int_compare:nNnT { #1 } = 1
6778     {
6779         \dim_gset:Nn \g_@@_blocks_ht_dim
6780         {
6781             \dim_max:nn
6782             \g_@@_blocks_ht_dim
6783             {
6784                 \box_ht:c
6785                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6786             }
6787         }
6788         \dim_gset:Nn \g_@@_blocks_dp_dim
6789         {

```

```

6790         \dim_max:nn
6791         \g_@@_blocks_dp_dim
6792         {
6793             \box_dp:c
6794             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6795         }
6796     }
6797 }
6798 \seq_gput_right:Nx \g_@@_blocks_seq
6799 {
6800     \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.

```

6801     {
6802         \exp_not:n { #3 } ,
6803         \l_@@_hpos_block_str ,
6804         \bool_if:NT \g_@@_rotate_c_bool { v-center }
6805     }
6806     {
6807         \box_use_drop:c
6808         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6809     }
6810 }
6811 \bool_set_false:N \g_@@_rotate_c_bool
6812 }

```

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.

```

6813 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
6814 {
6815     \seq_gput_right:Nx \g_@@_blocks_seq
6816     {
6817         \l_tmpa_tl
6818         { \exp_not:n { #3 } }
6819         {
6820             \bool_if:NTF \l_@@_tabular_bool
6821             {
6822                 \group_begin:
6823                 \bool_if:NF \l_@@_respect_arraystretch_bool
6824                 { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6825                 \exp_not:n
6826                 {
6827                     \dim_zero:N \extrarowheight
6828                     #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).

```

6829                 \bool_if:NT \g_@@_rotate_bool
6830                 { \str_set:Nn \l_@@_hpos_block_str c }
6831                 \use:x
6832                 {
6833                     \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
6834                     { @ { } \l_@@_hpos_block_str @ { } }
6835                 }
6836                 #5
6837                 \end { tabular }

```

```

6838     }
6839     \group_end:
6840 }
6841 {
6842     \group_begin:
6843     \bool_if:NF \l_@@_respect_arraystretch_bool
6844     { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6845     \exp_not:n
6846     {
6847         \dim_zero:N \extrarowheight
6848         #4
6849         \bool_if:NT \g_@@_rotate_bool
6850         { \str_set:Nn \l_@@_hpos_block_str c }
6851         \c_math_toggle_token
6852         \use:x
6853         {
6854             \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
6855             { @ { } \l_@@_hpos_block_str @ { } }
6856         }
6857         #5
6858         \end { array }
6859         \c_math_toggle_token
6860     }
6861     \group_end:
6862 }
6863 }
6864 }
6865 }

```

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).

```

6866 \keys_define:nn { NiceMatrix / Block / SecondPass }
6867 {
6868     tikz .code:n =
6869         \IfPackageLoadedTF { tikz }
6870         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6871         { \@@_error:n { tikz-key-without~tikz } } ,
6872     tikz .value_required:n = true ,
6873     fill .code:n =
6874         \tl_set_rescan:Nnn
6875         \l_@@_fill_tl
6876         { \char_set_catcode_other:N ! }
6877         { #1 } ,
6878     fill .value_required:n = true ,
6879     draw .code:n =
6880         \tl_set_rescan:Nnn
6881         \l_@@_draw_tl
6882         { \char_set_catcode_other:N ! }
6883         { #1 } ,
6884     draw .default:n = default ,
6885     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6886     rounded-corners .default:n = 4 pt ,
6887     color .code:n =
6888         \@@_color:n { #1 }
6889         \tl_set_rescan:Nnn
6890         \l_@@_draw_tl
6891         { \char_set_catcode_other:N ! }
6892         { #1 } ,
6893     color .value_required:n = true ,
6894     borders .clist_set:N = \l_@@_borders_clist ,
6895     borders .value_required:n = true ,

```

```

6896   hvlines .meta:n = { vlines , hlines } ,
6897   vlines .bool_set:N = \l_@@_vlines_block_bool ,
6898   vlines .default:n = true ,
6899   hlines .bool_set:N = \l_@@_hlines_block_bool ,
6900   hlines .default:n = true ,
6901   line-width .dim_set:N = \l_@@_line_width_dim ,
6902   line-width .value_required:n = true ,
6903   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6904   l .value_forbidden:n = true ,
6905   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6906   r .value_forbidden:n = true ,
6907   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6908   c .value_forbidden:n = true ,
6909   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6910           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6911   L .value_forbidden:n = true ,
6912   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6913           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6914   R .value_forbidden:n = true ,
6915   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6916           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6917   C .value_forbidden:n = true ,
6918   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6919   t .value_forbidden:n = true ,
6920   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6921   T .value_forbidden:n = true ,
6922   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6923   b .value_forbidden:n = true ,
6924   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6925   B .value_forbidden:n = true ,
6926   v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
6927   v-center .value_forbidden:n = true ,
6928   name .tl_set:N = \l_@@_block_name_str ,
6929   name .value_required:n = true ,
6930   name .initial:n = ,
6931   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6932   respect-arraystretch .default:n = true ,
6933   transparent .bool_set:N = \l_@@_transparent_bool ,
6934   transparent .default:n = true ,
6935   transparent .initial:n = false ,
6936   unknown .code:n = \@@_error:n { Unknown-key-for-Block }
6937 }

```

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.

```

6938 \cs_new_protected:Npn \@@_draw_blocks:
6939 {
6940   \cs_set_eq:NN \ialign \@@_old_ialign:
6941   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
6942 }
6943 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6944 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6945   \int_zero_new:N \l_@@_last_row_int
6946   \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.

```

6947 \int_compare:nNnTF { #3 } > { 99 }
6948 { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
6949 { \int_set:Nn \l_@@_last_row_int { #3 } }
6950 \int_compare:nNnTF { #4 } > { 99 }
6951 { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
6952 { \int_set:Nn \l_@@_last_col_int { #4 } }
6953 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6954 {
6955 \int_compare:nTF
6956 { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6957 {
6958 \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
6959 @@_msg_redirect_name:nn { Block~too~large~2 } { none }
6960 @@_msg_redirect_name:nn { columns~not~used } { none }
6961 }
6962 { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
6963 }
6964 {
6965 \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6966 { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
6967 { @@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6968 }
6969 }

```

The following command \@@_Block_v:nnnnnn will actually draw the block. #1 is the first row of the block; #2 is the first column of the block; #3 is the last row of the block; #4 is the last column of the block; #5 is a list of *key=value* options; #6 is the label

```

6970 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6971 {

```

The group is for the keys.

```

6972 \group_begin:
6973 \int_compare:nNnT { #1 } = { #3 }
6974 { \str_set:Nn \l_@@_vpos_of_block_str { t } }
6975 \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
6976 \bool_if:NT \l_@@_vlines_block_bool
6977 {
6978 \tl_gput_right:Nx \g_nicematrix_code_after_tl
6979 {
6980 \@@_vlines_block:nnn
6981 { \exp_not:n { #5 } }
6982 { #1 - #2 }
6983 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6984 }
6985 }
6986 \bool_if:NT \l_@@_hlines_block_bool
6987 {
6988 \tl_gput_right:Nx \g_nicematrix_code_after_tl
6989 {
6990 \@@_hlines_block:nnn
6991 { \exp_not:n { #5 } }
6992 { #1 - #2 }
6993 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6994 }
6995 }
6996 \bool_if:nF
6997 {
6998 \l_@@_transparent_bool
6999 || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7000 }
7001 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7002     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7003     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7004 }

7005 \bool_lazy_and:nnT
7006 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
7007 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7008 { \@@_error:n { hlines-with-color } }

7009 \tl_if_empty:NF \l_@@_draw_tl
7010 {
7011     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7012     {
7013         \@@_stroke_block:nnn
7014         { \exp_not:n { #5 } }
7015         { #1 - #2 }
7016         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7017     }
7018     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7019     { { #1 } { #2 } { #3 } { #4 } }
7020 }

7021 \clist_if_empty:NF \l_@@_borders_clist
7022 {
7023     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7024     {
7025         \@@_stroke_borders_block:nnn
7026         { \exp_not:n { #5 } }
7027         { #1 - #2 }
7028         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7029     }
7030 }

7031 \tl_if_empty:NF \l_@@_fill_tl
7032 {
7033     \tl_gput_right:Nx \g_@@_pre_code_before_tl
7034     {
7035         \exp_not:N \roundedrectanglecolor
7036         \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7037             { \l_@@_fill_tl }
7038             { { \l_@@_fill_tl } }
7039             { #1 - #2 }
7040             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7041             { \dim_use:N \l_@@_rounded_corners_dim }
7042         ]
7043     }
7044 }

7045 \seq_if_empty:NF \l_@@_tikz_seq
7046 {
7047     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7048     {
7049         \@@_block_tikz:nnnn
7050         { #1 }
7051         { #2 }
7052         { \int_use:N \l_@@_last_row_int }
7053         { \int_use:N \l_@@_last_col_int }
7054         { \seq_use:Nn \l_@@_tikz_seq { , } }
7055     }
7056 }

7056 \cs_set_protected_nopar:Npn \diagbox ##1 ##2

```

```

7057 {
7058   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7059   {
7060     \@@_actually_diagbox:nnnnnn
7061     { #1 }
7062     { #2 }
7063     { \int_use:N \l_@@_last_row_int }
7064     { \int_use:N \l_@@_last_col_int }
7065     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7066   }
7067 }

7068 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7069 \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.

```

7070 \pgfpicture
7071 \pgfrememberpicturepositiononpagetrue
7072 \pgf@relevantforpicturesizefalse
7073 \@@_qpoint:n { row - #1 }
7074 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7075 \@@_qpoint:n { col - #2 }
7076 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7077 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7078 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7079 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7080 \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.

```

7081 \@@_pgf_rect_node:nnnnn
7082 { \@@_env: - #1 - #2 - block }
7083 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7084 \str_if_empty:NF \l_@@_block_name_str
7085 {
7086   \pgfnodealias
7087   { \@@_env: - \l_@@_block_name_str }
7088   { \@@_env: - #1 - #2 - block }
7089   \str_if_empty:NF \l_@@_name_str
7090   {
7091     \pgfnodealias
7092     { \l_@@_name_str - \l_@@_block_name_str }

```

```

7093         { \@@_env: - #1 - #2 - block }
7094     }
7095 }

```

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.

```

7096 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7097 {
7098     \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.

```

7099     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7100     {

```

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.

```

7101         \cs_if_exist:cT
7102         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7103         {
7104             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7105             {
7106                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7107                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7108             }
7109         }
7110     }

```

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.

```

7111     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7112     {
7113         \@@_qpoint:n { col - #2 }
7114         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7115     }
7116     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7117     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7118     {
7119         \cs_if_exist:cT
7120         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7121         {
7122             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7123             {
7124                 \pgfpointanchor
7125                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7126                 { east }
7127                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7128             }
7129         }
7130     }
7131     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7132     {
7133         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7134         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7135     }
7136     \@@_pgf_rect_node:nnnnn
7137     { \@@_env: - #1 - #2 - block - short }
7138     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7139 }

```

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.

```

7140 \bool_if:NT \l_@@_medium_nodes_bool
7141 {
7142   \@@_pgf_rect_node:nnn
7143   { \@@_env: - #1 - #2 - block - medium }
7144   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7145   {
7146     \pgfpointanchor
7147     { \@@_env:
7148       - \int_use:N \l_@@_last_row_int
7149       - \int_use:N \l_@@_last_col_int - medium
7150     }
7151     { south-east }
7152   }
7153 }

```

Now, we will put the label of the block.

```

7154 \bool_lazy_any:nTF
7155 {
7156   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7157   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7158   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7159 }

```

```

7160 {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7161   \int_compare:nNnT { #2 } = 0
7162   { \str_set:Nn \l_@@_hpos_block_str r }
7163   \bool_if:nT \g_@@_last_col_found_bool
7164   {
7165     \int_compare:nNnT { #2 } = \g_@@_col_total_int
7166     { \str_set:Nn \l_@@_hpos_block_str l }
7167   }
7168   \tl_set:Nx \l_tmpa_tl
7169   {
7170     \str_case:Vn \l_@@_vpos_of_block_str
7171     {
7172       c {
7173         \str_case:Vn \l_@@_hpos_block_str
7174         {
7175           c { center }
7176           l { west }
7177           r { east }
7178         }
7179       }
7180     }
7181     T {
7182       \str_case:Vn \l_@@_hpos_block_str
7183       {
7184         c { north }
7185         l { north-west }
7186         r { north-east }
7187       }
7188     }
7189     B {
7190       \str_case:Vn \l_@@_hpos_block_str
7191       {
7192         c { south }
7193         l { south-west }
7194         r { south-east }
7195       }
7196     }
7197   }

```

```

7198         }
7199     }
7200 }
7201 \pgftransformshift
7202 {
7203     \pgfpointanchor
7204     {
7205         \@@_env: - #1 - #2 - block
7206         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7207     }
7208     { \l_tmpa_tl }
7209 }
7210 \pgfset
7211 {
7212     inner~xsep = \c_zero_dim ,
7213     inner~ysep = \l_@@_block_ysep_dim
7214 }
7215 \pgfnode
7216 { rectangle }
7217 { \l_tmpa_tl }
7218 { \box_use_drop:N \l_@@_cell_box } { } { }
7219 }
7220 {
7221     \pgfextracty \l_tmpa_dim
7222     {
7223         \@@_qpoint:n
7224         {
7225             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7226             - base
7227         }
7228     }
7229     \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.

```

7230 \pgfpointanchor
7231 {
7232     \@@_env: - #1 - #2 - block
7233     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7234 }
7235 {
7236     \str_case:Vn \l_@@_hpos_block_str
7237     {
7238         c { center }
7239         l { west }
7240         r { east }
7241     }
7242 }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

7243 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7244 \pgfset { inner~sep = \c_zero_dim }
7245 \pgfnode
7246 { rectangle }
7247 {
7248     \str_case:Vn \l_@@_hpos_block_str
7249     {
7250         c { base }
7251         l { base~west }
7252         r { base~east }
7253     }
7254 }
7255 { \box_use_drop:N \l_@@_cell_box } { } { }
7256 }

```

```

7257 \endpgfpicture
7258 \group_end:
7259 }

```

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).

```

7260 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7261 {
7262   \group_begin:
7263   \tl_clear:N \l_@@_draw_tl
7264   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7265   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7266   \pgfpicture
7267   \pgfrememberpicturepositiononpagetrue
7268   \pgf@relevantforpicturesizefalse
7269   \tl_if_empty:NF \l_@@_draw_tl
7270   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7271     \str_if_eq:VnTF \l_@@_draw_tl { default }
7272     { \CT@arc@ }
7273     { \@@_color:V \l_@@_draw_tl }
7274   }
7275   \pgfsetcornersarced
7276   {
7277     \pgfpoint
7278     { \dim_use:N \l_@@_rounded_corners_dim }
7279     { \dim_use:N \l_@@_rounded_corners_dim }
7280   }
7281   \@@_cut_on_hyphen:w #2 \q_stop
7282   \bool_lazy_and:nnT
7283   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7284   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7285   {
7286     \@@_qpoint:n { row - \l_tmpa_tl }
7287     \dim_set:Nn \l_tmpb_dim { \pgf@y }
7288     \@@_qpoint:n { col - \l_tmpb_tl }
7289     \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
7290     \@@_cut_on_hyphen:w #3 \q_stop
7291     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7292     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7293     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7294     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7295     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7296     \dim_set:Nn \l_tmpa_dim { \pgf@y }
7297     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7298     \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
7299     \pgfpathrectanglecorners
7300     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7301     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7302     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7303     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7304     { \pgfusepathqstroke }
7305     { \pgfusepath { stroke } }
7306   }
7307   \endpgfpicture
7308   \group_end:
7309 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7310 \keys_define:nn { NiceMatrix / BlockStroke }

```

```

7311 {
7312   color .tl_set:N = \l_@@_draw_tl ,
7313   draw .code:n =
7314     \exp_args:Nx \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7315   draw .default:n = default ,
7316   line-width .dim_set:N = \l_@@_line_width_dim ,
7317   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7318   rounded-corners .default:n = 4 pt
7319 }

```

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).

```

7320 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7321 {
7322   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7323   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7324   \@@_cut_on_hyphen:w #2 \q_stop
7325   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7326   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7327   \@@_cut_on_hyphen:w #3 \q_stop
7328   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7329   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7330   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7331   {
7332     \use:x
7333     {
7334       \@@_vline:n
7335       {
7336         position = ##1 ,
7337         start = \l_@@_tmpc_tl ,
7338         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7339         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7340       }
7341     }
7342   }
7343 }
7344 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7345 {
7346   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7347   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7348   \@@_cut_on_hyphen:w #2 \q_stop
7349   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7350   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7351   \@@_cut_on_hyphen:w #3 \q_stop
7352   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7353   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7354   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7355   {
7356     \use:x
7357     {
7358       \@@_hline:n
7359       {
7360         position = ##1 ,
7361         start = \l_@@_tmpd_tl ,
7362         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7363         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7364       }
7365     }
7366   }
7367 }

```

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).

```

7368 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7369 {
7370   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7371   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7372   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7373     { \@@_error:n { borders~forbidden } }
7374     {
7375       \tl_clear_new:N \l_@@_borders_tikz_tl
7376       \keys_set:nV
7377         { NiceMatrix / OnlyForTikzInBorders }
7378         \l_@@_borders_clist
7379       \@@_cut_on_hyphen:w #2 \q_stop
7380       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7381       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7382       \@@_cut_on_hyphen:w #3 \q_stop
7383       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7384       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7385       \@@_stroke_borders_block_i:
7386     }
7387 }
7388 \hook_gput_code:nnn { begindocument } { . }
7389 {
7390   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7391   {
7392     \c_@@_pgfortikzpicture_tl
7393     \@@_stroke_borders_block_ii:
7394     \c_@@_endpgfortikzpicture_tl
7395   }
7396 }
7397 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7398 {
7399   \pgfrememberpicturepositiononpagetrue
7400   \pgf@relevantforpicturesizefalse
7401   \CT@arc@
7402   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7403   \clist_if_in:NnT \l_@@_borders_clist { right }
7404     { \@@_stroke_vertical:n \l_tmpb_tl }
7405   \clist_if_in:NnT \l_@@_borders_clist { left }
7406     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7407   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7408     { \@@_stroke_horizontal:n \l_tmpa_tl }
7409   \clist_if_in:NnT \l_@@_borders_clist { top }
7410     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7411 }
7412 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7413 {
7414   tikz .code:n =
7415     \cs_if_exist:NTF \tikzpicture
7416       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7417       { \@@_error:n { tikz-in~borders~without-tikz } } ,
7418   tikz .value_required:n = true ,
7419   top .code:n = ,
7420   bottom .code:n = ,
7421   left .code:n = ,
7422   right .code:n = ,
7423   unknown .code:n = \@@_error:n { bad~border }
7424 }

```

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).

```

7425 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7426 {
7427   \@@_qpoint:n \l_@@_tmpc_tl
7428   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7429   \@@_qpoint:n \l_tmpa_tl
7430   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7431   \@@_qpoint:n { #1 }
7432   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7433   {
7434     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7435     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7436     \pgfusepathqstroke
7437   }
7438   {
7439     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7440     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7441   }
7442 }

```

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).

```

7443 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7444 {
7445   \@@_qpoint:n \l_@@_tmpd_tl
7446   \clist_if_in:NnTF \l_@@_borders_clist { left }
7447   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7448   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7449   \@@_qpoint:n \l_tmpb_tl
7450   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7451   \@@_qpoint:n { #1 }
7452   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7453   {
7454     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7455     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7456     \pgfusepathqstroke
7457   }
7458   {
7459     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7460     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7461   }
7462 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

7463 \keys_define:nn { NiceMatrix / BlockBorders }
7464 {
7465   borders .clist_set:N = \l_@@_borders_clist ,
7466   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7467   rounded-corners .default:n = 4 pt ,
7468   line-width .dim_set:N = \l_@@_line_width_dim ,
7469 }

```

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.

```

7470 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7471 {
7472   \begin { tikzpicture }
7473   \clist_map_inline:nn { #5 }
7474   {
7475     \path [ ##1 ]
7476     ( #1 -| #2 )
7477     rectangle

```

```

7478      ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7479    }
7480    \end { tikzpicture }
7481  }

```

27 How to draw the dotted lines transparently

```

7482 \cs_set_protected:Npn \@@_renew_matrix:
7483 {
7484   \RenewDocumentEnvironment { pmatrix } { } { }
7485   { \pNiceMatrix }
7486   { \endpNiceMatrix }
7487   \RenewDocumentEnvironment { vmatrix } { } { }
7488   { \vNiceMatrix }
7489   { \endvNiceMatrix }
7490   \RenewDocumentEnvironment { Vmatrix } { } { }
7491   { \VNiceMatrix }
7492   { \endVNiceMatrix }
7493   \RenewDocumentEnvironment { bmatrix } { } { }
7494   { \bNiceMatrix }
7495   { \endbNiceMatrix }
7496   \RenewDocumentEnvironment { Bmatrix } { } { }
7497   { \BNiceMatrix }
7498   { \endBNiceMatrix }
7499 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7500 \keys_define:nn { NiceMatrix / Auto }
7501 {
7502   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7503   columns-type .value_required:n = true ,
7504   l .meta:n = { columns-type = l } ,
7505   r .meta:n = { columns-type = r } ,
7506   c .meta:n = { columns-type = c } ,
7507   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7508   delimiters / color .value_required:n = true ,
7509   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7510   delimiters / max-width .default:n = true ,
7511   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7512   delimiters .value_required:n = true ,
7513   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7514   rounded-corners .default:n = 4 pt
7515 }
7516 \NewDocumentCommand \AutoNiceMatrixWithDelims
7517 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7518 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7519 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7520 {

```

The group is for the protection of the keys.

```

7521   \group_begin:
7522   \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl

```

We nullify the command `\@@_transform_preamble_i:` because we will provide a preamble which is yet transformed (by using `\l_@@_columns_type_tl` which is yet `nicematrix`-ready).

```

7523   \bool_set_false:N \l_@@_preamble_bool

```

```

7524 \use:x
7525 {
7526   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7527   { * { #4 } { \exp_not:V \l_@@_columns_type_tl } }
7528   [ \exp_not:V \l_tmpa_tl ]
7529 }
7530 \int_compare:nNnT \l_@@_first_row_int = 0
7531 {
7532   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7533   \prg_replicate:nn { #4 - 1 } { & }
7534   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7535 }
7536 \prg_replicate:nn { #3 }
7537 {
7538   \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).

```

7539   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7540   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7541 }
7542 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7543 {
7544   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7545   \prg_replicate:nn { #4 - 1 } { & }
7546   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7547 }
7548 \end { NiceArrayWithDelims }
7549 \group_end:
7550 }
7551 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7552 {
7553   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7554   {
7555     \bool_gset_true:N \g_@@_delims_bool
7556     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7557     \AutoNiceMatrixWithDelims { #2 } { #3 }
7558   }
7559 }
7560 \@@_define_com:nnn p ( )
7561 \@@_define_com:nnn b [ ]
7562 \@@_define_com:nnn v | |
7563 \@@_define_com:nnn V \ | \ |
7564 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

7565 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7566 {
7567   \group_begin:
7568   \bool_gset_false:N \g_@@_delims_bool
7569   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7570   \group_end:
7571 }

```

29 The redefinition of the command \dotfill

```

7572 \cs_set_eq:NN \@@_old_dotfill \dotfill

```

```

7573 \cs_new_protected:Npn \@@_dotfill:
7574 {

```

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}`).

```

7575 \@@_old_dotfill
7576 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7577 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

7578 \cs_new_protected:Npn \@@_dotfill_i:
7579 { \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.

```

7580 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7581 {
7582   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7583   {
7584     \@@_actually_diagbox:nnnnnn
7585     { \int_use:N \c@iRow }
7586     { \int_use:N \c@jCol }
7587     { \int_use:N \c@iRow }
7588     { \int_use:N \c@jCol }
7589     { \exp_not:n { #1 } }
7590     { \exp_not:n { #2 } }
7591   }

```

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`.

```

7592 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7593 {
7594   { \int_use:N \c@iRow }
7595   { \int_use:N \c@jCol }
7596   { \int_use:N \c@iRow }
7597   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7598   { }
7599 }
7600 }

```

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.

```

7601 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7602 {
7603   \pgfpicture
7604   \pgf@relevantforpicturesizefalse
7605   \pgfrememberpicturepositiononpagetrue
7606   \@@_qpoint:n { row - #1 }
7607   \dim_set_eq:NN \l_tmpa_dim \pgf@y
7608   \@@_qpoint:n { col - #2 }
7609   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7610   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7611   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7612   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y

```

```

7613 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7614 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7615 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7616 {

```

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.

```

7617 \CT@arc@
7618 \pgfsetroundcap
7619 \pgfusepathqstroke
7620 }
7621 \pgfset { inner~sep = 1 pt }
7622 \pgfscope
7623 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7624 \pgfnode { rectangle } { south-west }
7625 {
7626 \begin { minipage } { 20 cm }
7627 \@@_math_toggle_token: #5 \@@_math_toggle_token:
7628 \end { minipage }
7629 }
7630 { }
7631 { }
7632 \endpgfscope
7633 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7634 \pgfnode { rectangle } { north-east }
7635 {
7636 \begin { minipage } { 20 cm }
7637 \raggedleft
7638 \@@_math_toggle_token: #6 \@@_math_toggle_token:
7639 \end { minipage }
7640 }
7641 { }
7642 { }
7643 \endpgfpicture
7644 }

```

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.

```

7645 \keys_define:nn { NiceMatrix }
7646 {
7647 \CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7648 \CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7649 }
7650 \keys_define:nn { NiceMatrix / CodeAfter }
7651 {
7652 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7653 sub-matrix .value_required:n = true ,
7654 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7655 delimiters / color .value_required:n = true ,
7656 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7657 rules .value_required:n = true ,
7658 unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
7659 }

```

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`.

```
7660 \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 `\`.

```
7661 \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`.

```
7662 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7663 {
7664   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7665   \@@_CodeAfter_iv:n
7666 }
```

We catch the argument of the command `\end` (in `#1`).

```
7667 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7668 {
```

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.

```
7669   \str_if_eq:eeTF \@currentenv { #1 }
7670   { \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`.

```
7671   {
7672     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7673     \@@_CodeAfter_ii:n
7674   }
7675 }
```

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.

```
7676 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7677 {
7678   \pgfpicture
7679   \pgfrememberpicturepositiononpagetrue
7680   \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.

```
7681   \@@_qpoint:n { row - 1 }
7682   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7683   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7684   \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).

```

7685 \bool_if:nTF { #3 }
7686 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7687 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7688 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7689 {
7690   \cs_if_exist:cT
7691   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7692   {
7693     \pgfpointanchor
7694     { \@@_env: - ##1 - #2 }
7695     { \bool_if:nTF { #3 } { west } { east } }
7696     \dim_set:Nn \l_tmpa_dim
7697     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7698   }
7699 }

```

Now we can put the delimiter with a node of PGF.

```

7700 \pgfset { inner~sep = \c_zero_dim }
7701 \dim_zero:N \nulldelimiterspace
7702 \pgftransformshift
7703 {
7704   \pgfpoint
7705   { \l_tmpa_dim }
7706   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7707 }
7708 \pgfnode
7709 { rectangle }
7710 { \bool_if:nTF { #3 } { east } { west } }
7711 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7712 \nullfont
7713 \c_math_toggle_token
7714 \@@_color:V \l_@@_delimiters_color_tl
7715 \bool_if:nTF { #3 } { \left #1 } { \left . }
7716 \vcenter
7717 {
7718   \nullfont
7719   \hrule \@height
7720   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7721   \@depth \c_zero_dim
7722   \@width \c_zero_dim
7723 }
7724 \bool_if:nTF { #3 } { \right . } { \right #1 }
7725 \c_math_toggle_token
7726 }
7727 { }
7728 { }
7729 \endpgfpicture
7730 }

```

33 The command `\SubMatrix`

```

7731 \keys_define:nn { NiceMatrix / sub-matrix }
7732 {
7733   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7734   extra-height .value_required:n = true ,
7735   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7736   left-xshift .value_required:n = true ,

```



```

7737 right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7738 right-xshift .value_required:n = true ,
7739 xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7740 xshift .value_required:n = true ,
7741 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7742 delimiters / color .value_required:n = true ,
7743 slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7744 slim .default:n = true ,
7745 hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7746 hlines .default:n = all ,
7747 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7748 vlines .default:n = all ,
7749 hvlines .meta:n = { hlines, vlines } ,
7750 hvlines .value_forbidden:n = true ,
7751 }
7752 \keys_define:nn { NiceMatrix }
7753 {
7754   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7755   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7756   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7757   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7758   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7759   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7760 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

7761 \keys_define:nn { NiceMatrix / SubMatrix }
7762 {
7763   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7764   delimiters / color .value_required:n = true ,
7765   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7766   hlines .default:n = all ,
7767   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7768   vlines .default:n = all ,
7769   hvlines .meta:n = { hlines, vlines } ,
7770   hvlines .value_forbidden:n = true ,
7771   name .code:n =
7772     \tl_if_empty:nTF { #1 }
7773     { \@@_error:n { Invalid-name } }
7774     {
7775       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7776       {
7777         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7778         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7779         {
7780           \str_set:Nn \l_@@_submatrix_name_str { #1 }
7781           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7782         }
7783       }
7784       { \@@_error:n { Invalid-name } }
7785     } ,
7786   name .value_required:n = true ,
7787   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7788   rules .value_required:n = true ,
7789   code .tl_set:N = \l_@@_code_tl ,
7790   code .value_required:n = true ,
7791   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
7792 }

7793 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7794 {
7795   \peek_remove_spaces:n

```

```

7796 {
7797   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7798   {
7799     \SubMatrix { #1 } { #2 } { #3 } { #4 }
7800     [
7801       delimiters / color = \l_@@_delimiters_color_tl ,
7802       hlines = \l_@@_submatrix_hlines_clist ,
7803       vlines = \l_@@_submatrix_vlines_clist ,
7804       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
7805       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
7806       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
7807       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
7808       #5
7809     ]
7810   }
7811   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7812 }
7813 }
7814 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
7815 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7816 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
7817 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
7818 {
7819   \seq_gput_right:Nx \g_@@_submatrix_seq
7820   {
We use \str_if_eq:nnTF because it is fully expandable.
7821     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
7822     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
7823     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
7824     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
7825   }
7826 }

```

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`.

```

7827 \hook_gput_code:nnn { begindocument } { . }
7828 {
7829   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
7830   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
7831   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
7832   {
7833     \peek_remove_spaces:n
7834     {
7835       \@@_sub_matrix:nnnnnnn
7836       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
7837     }
7838   }
7839 }

```

The following macro will compute $\backslash l_{@@_first_i_tl}$, $\backslash l_{@@_first_j_tl}$, $\backslash l_{@@_last_i_tl}$ and $\backslash l_{@@_last_j_tl}$ from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

7840 \NewDocumentCommand \@@_compute_i_j:nn
7841 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7842 { \@@_compute_i_j:nnnn #1 #2 }

7843 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
7844 {
7845   \tl_set:Nn \l_@@_first_i_tl { #1 }
7846   \tl_set:Nn \l_@@_first_j_tl { #2 }
7847   \tl_set:Nn \l_@@_last_i_tl { #3 }
7848   \tl_set:Nn \l_@@_last_j_tl { #4 }
7849   \tl_if_eq:NnT \l_@@_first_i_tl { last }
7850     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
7851   \tl_if_eq:NnT \l_@@_first_j_tl { last }
7852     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
7853   \tl_if_eq:NnT \l_@@_last_i_tl { last }
7854     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
7855   \tl_if_eq:NnT \l_@@_last_j_tl { last }
7856     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
7857 }

7858 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
7859 {
7860   \group_begin:

```

The four following token lists correspond to the position of the \backslash SubMatrix.

```

7861 \@@_compute_i_j:nn { #2 } { #3 }
7862 % added 6.19b
7863 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
7864 { \cs_set:Npn \arraystretch { 1 } }
7865 \bool_lazy_or:nnTF
7866 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7867 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7868 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
7869 {
7870   \str_clear_new:N \l_@@_submatrix_name_str
7871   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
7872   \pgfpicture
7873   \pgfrememberpicturerepositiononpagetrue
7874   \pgf@relevantforpicturesizefalse
7875   \pgfset { inner~sep = \c_zero_dim }
7876   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7877   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \backslash int_step_inline:nnn is provided by currfication.

```

7878 \bool_if:NTF \l_@@_submatrix_slim_bool
7879 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
7880 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
7881 {
7882   \cs_if_exist:cT
7883     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7884     {
7885       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7886       \dim_set:Nn \l_@@_x_initial_dim
7887         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7888     }
7889   \cs_if_exist:cT
7890     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7891     {
7892       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7893       \dim_set:Nn \l_@@_x_final_dim
7894         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7895     }

```

```

7896     }
7897     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7898     { \@@_error:nn { Impossible~delimiter } { left } }
7899     {
7900         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7901         { \@@_error:nn { Impossible~delimiter } { right } }
7902         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7903     }
7904     \endpgfpicture
7905 }
7906 \group_end:
7907 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7908 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7909 {
7910     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7911     \dim_set:Nn \l_@@_y_initial_dim
7912     {
7913         \fp_to_dim:n
7914         {
7915             \pgf@y
7916             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
7917         }
7918     } % modified 6.13c
7919     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7920     \dim_set:Nn \l_@@_y_final_dim
7921     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
7922     % modified 6.13c
7923     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7924     {
7925         \cs_if_exist:cT
7926         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7927         {
7928             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7929             \dim_set:Nn \l_@@_y_initial_dim
7930             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7931         }
7932         \cs_if_exist:cT
7933         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7934         {
7935             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7936             \dim_set:Nn \l_@@_y_final_dim
7937             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7938         }
7939     }
7940     \dim_set:Nn \l_tmpa_dim
7941     {
7942         \l_@@_y_initial_dim - \l_@@_y_final_dim +
7943         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7944     }
7945     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7946 \group_begin:
7947 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7948 \@@_set_CT@arc@:V \l_@@_rules_color_tl
7949 \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`.

```

7950 \seq_map_inline:Nn \g_@@_cols_vlism_seq
7951 {
7952   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7953   {
7954     \int_compare:nNnT
7955       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
7956     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

7957       \@@_qpoint:n { col - ##1 }
7958       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7959       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7960       \pgfusepathqstroke
7961     }
7962   }
7963 }

```

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.

```

7964 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7965 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7966 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7967 {
7968   \bool_lazy_and:nnTF
7969     { \int_compare_p:nNn { ##1 } > 0 }
7970     {
7971       \int_compare_p:nNn
7972         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7973     {
7974       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7975       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7976       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7977       \pgfusepathqstroke
7978     }
7979     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
7980 }

```

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.

```

7981 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7982 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7983 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7984 {
7985   \bool_lazy_and:nnTF
7986     { \int_compare_p:nNn { ##1 } > 0 }
7987     {
7988       \int_compare_p:nNn
7989         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7990     {
7991       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7992 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

7993 \dim_set:Nn \l_tmpa_dim
7994 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7995 \str_case:nn { #1 }
7996 {
7997   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7998   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7999   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8000 }
8001 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8002         \dim_set:Nn \l_tmpb_dim
8003         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8004         \str_case:nn { #2 }
8005         {
8006             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8007             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8008             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8009         }
8010         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8011         \pgfusepathqstroke
8012         \group_end:
8013     }
8014     { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8015 }

```

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).

```

8016     \str_if_empty:NF \l_@@_submatrix_name_str
8017     {
8018         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8019         \l_@@_x_initial_dim \l_@@_y_initial_dim
8020         \l_@@_x_final_dim \l_@@_y_final_dim
8021     }
8022     \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`.

```

8023     \begin { pgfscope }
8024     \pgftransformshift
8025     {
8026         \pgfpoint
8027         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8028         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8029     }
8030     \str_if_empty:NTF \l_@@_submatrix_name_str
8031     { \@@_node_left:nn #1 { } }
8032     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8033     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8034     \pgftransformshift
8035     {
8036         \pgfpoint
8037         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8038         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8039     }
8040     \str_if_empty:NTF \l_@@_submatrix_name_str
8041     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8042     {
8043         \@@_node_right:nnnn #2
8044         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8045     }
8046     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8047     \flag_clear_new:n { nicematrix }
8048     \l_@@_code_tl
8049 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the

number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```
8050 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor
```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpntanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```
8051 \cs_new_protected:Npn \@@_pgfpntanchor:n #1
8052 {
8053   \use:e
8054   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
8055 }
```

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 -.

```
8056 \cs_new:Npn \@@_pgfpntanchor_i:nn #1 #2
8057 { #1 { \@@_pgfpntanchor_ii:w #2 - \q_stop } }
```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```
8058 \tl_const:Nn \c_@@_integers_alist_tl
8059 {
8060   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8061   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8062   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8063   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8064 }

8065 \cs_new:Npn \@@_pgfpntanchor_ii:w #1-#2\q_stop
8066 {
```

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`.

```
8067   \tl_if_empty:nTF { #2 }
8068   {
8069     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8070     {
8071       \flag_raise:n { nicematrix }
8072       \int_if_even:nTF { \flag_height:n { nicematrix } }
8073       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8074       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8075     }
8076     { #1 }
8077   }
```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```
8078   { \@@_pgfpntanchor_iii:w { #1 } #2 }
8079 }
```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpntanchor_i:nn`).

```
8080 \cs_new:Npn \@@_pgfpntanchor_iii:w #1 #2 -
```

```

8081 {
8082   \str_case:nnF { #1 }
8083   {
8084     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8085     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8086   }

```

Now the case of a node of the form $i-j$.

```

8087 {
8088   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8089   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8090 }
8091 }

```

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`).

```

8092 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8093 {
8094   \pgfnode
8095   { rectangle }
8096   { east }
8097   {
8098     \nullfont
8099     \c_math_toggle_token
8100     \@@_color:V \l_@@_delimiters_color_tl
8101     \left #1
8102     \vcenter
8103     {
8104       \nullfont
8105       \hrule \@height \l_tmpa_dim
8106       \@depth \c_zero_dim
8107       \@width \c_zero_dim
8108     }
8109     \right .
8110     \c_math_toggle_token
8111   }
8112   { #2 }
8113   { }
8114 }

```

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.

```

8115 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8116 {
8117   \pgfnode
8118   { rectangle }
8119   { west }
8120   {
8121     \nullfont
8122     \c_math_toggle_token
8123     \@@_color:V \l_@@_delimiters_color_tl
8124     \left .
8125     \vcenter
8126     {
8127       \nullfont
8128       \hrule \@height \l_tmpa_dim
8129       \@depth \c_zero_dim
8130       \@width \c_zero_dim
8131     }
8132     \right #1
8133     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }

```



```

8134      ^ { \smash { #4 } }
8135      \c_math_toggle_token
8136    }
8137    { #2 }
8138    { }
8139  }

```

34 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

8140 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8141 {
8142   \peek_remove_spaces:n
8143   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8144 }

8145 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8146 {
8147   \peek_remove_spaces:n
8148   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8149 }

8150 \keys_define:nn { NiceMatrix / Brace }
8151 {
8152   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8153   left-shorten .default:n = true ,
8154   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8155   shorten .meta:n = { left-shorten , right-shorten } ,
8156   right-shorten .default:n = true ,
8157   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8158   yshift .value_required:n = true ,
8159   yshift .initial:n = \c_zero_dim ,
8160   color .tl_set:N = \l_tmpa_tl ,
8161   color .value_required:n = true ,
8162   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8163 }

```

#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`.

```

8164 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8165 {
8166   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8167 \@@_compute_i_j:nn { #1 } { #2 }
8168 \bool_lazy_or:nnTF
8169 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8170 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8171 {
8172   \str_if_eq:nnTF { #5 } { under }
8173   { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8174   { \@@_error:nn { Construct~too~large } { \OverBrace } }
8175 }
8176 {
8177   \tl_clear:N \l_tmpa_tl
8178   \keys_set:nn { NiceMatrix / Brace } { #4 }
8179   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8180   \pgfpicture

```

```

8181 \pgfrememberpicturepositiononpagetrue
8182 \pgf@relevantforpicturesizefalse
8183 \bool_if:NT \l_@@_brace_left_shorten_bool
8184 {
8185     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8186     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8187     {
8188         \cs_if_exist:cT
8189         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8190         {
8191             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8192             \dim_set:Nn \l_@@_x_initial_dim
8193             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8194         }
8195     }
8196 }
8197 \bool_lazy_or:nnT
8198 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8199 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8200 {
8201     \@@_qpoint:n { col - \l_@@_first_j_tl }
8202     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8203 }
8204 \bool_if:NT \l_@@_brace_right_shorten_bool
8205 {
8206     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8207     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8208     {
8209         \cs_if_exist:cT
8210         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8211         {
8212             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8213             \dim_set:Nn \l_@@_x_final_dim
8214             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8215         }
8216     }
8217 }
8218 \bool_lazy_or:nnT
8219 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8220 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8221 {
8222     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8223     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8224 }
8225 \pgfset { inner-sep = \c_zero_dim }
8226 \str_if_eq:nnTF { #5 } { under }
8227 { \@@_underbrace_i:n { #3 } }
8228 { \@@_overbrace_i:n { #3 } }
8229 \endpgfpicture
8230 }
8231 \group_end:
8232 }

```

The argument is the text to put above the brace.

```

8233 \cs_new_protected:Npn \@@_overbrace_i:n #1
8234 {
8235     \@@_qpoint:n { row - \l_@@_first_i_tl }
8236     \pgftransformshift
8237     {
8238         \pgfpoint
8239         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8240         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8241     }
8242     \pgfnode

```

```

8243 { rectangle }
8244 { south }
8245 {
8246   \vbox_top:n
8247   {
8248     \group_begin:
8249     \everycr { }
8250     \halign
8251     {
8252       \hfil ## \hfil \crrc
8253       \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8254       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8255       \c_math_toggle_token
8256       \overbrace
8257       {
8258         \hbox_to_wd:nn
8259         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8260         { }
8261       }
8262       \c_math_toggle_token
8263       \cr
8264     }
8265     \group_end:
8266   }
8267 }
8268 { }
8269 { }
8270 }

```

The argument is the text to put under the brace.

```

8271 \cs_new_protected:Npn \@@_underbrace_i:n #1
8272 {
8273   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8274   \pgftransformshift
8275   {
8276     \pgfpoint
8277     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8278     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8279   }
8280   \pgfnode
8281   { rectangle }
8282   { north }
8283   {
8284     \group_begin:
8285     \everycr { }
8286     \vbox:n
8287     {
8288       \halign
8289       {
8290         \hfil ## \hfil \crrc
8291         \c_math_toggle_token
8292         \underbrace
8293         {
8294           \hbox_to_wd:nn
8295           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8296           { }
8297         }
8298         \c_math_toggle_token
8299         \cr
8300         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8301         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8302       }
8303     }
8304     \group_end:

```

```

8305     }
8306     { }
8307     { }
8308 }

```

35 The command \ShowCellNames

```

8309 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8310 {
8311   \dim_zero_new:N \g_@@_tmpc_dim
8312   \dim_zero_new:N \g_@@_tmpd_dim
8313   \dim_zero_new:N \g_@@_tmpe_dim
8314   \int_step_inline:nn \c@iRow
8315   {
8316     \begin { pgfpicture }
8317     \@@_qpoint:n { row - ##1 }
8318     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8319     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8320     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8321     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8322     \bool_if:NTF \l_@@_in_code_after_bool
8323     \end { pgfpicture }
8324     \int_step_inline:nn \c@jCol
8325     {
8326       \hbox_set:Nn \l_tmpa_box
8327       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8328       \begin { pgfpicture }
8329       \@@_qpoint:n { col - #####1 }
8330       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8331       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8332       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8333       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8334       \endpgfpicture
8335       \end { pgfpicture }
8336       \fp_set:Nn \l_tmpa_fp
8337       {
8338         \fp_min:nn
8339         {
8340           \fp_min:nn
8341           { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8342           { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8343         }
8344         { 1.0 }
8345       }
8346       \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8347       \pgfpicture
8348       \pgfrememberpicturepositiononpagetrue
8349       \pgf@relevantforpicturesizefalse
8350       \pgftransformshift
8351       {
8352         \pgfpoint
8353         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8354         { \dim_use:N \g_tmpa_dim }
8355       }
8356       \pgfnode
8357       { rectangle }
8358       { center }
8359       { \box_use:N \l_tmpa_box }
8360       { }

```

```

8361         { }
8362     \endpgfpicture
8363 }
8364 }
8365 }
8366 \NewDocumentCommand \@@_ShowCellNames { }
8367 {
8368     \bool_if:NT \l_@@_in_code_after_bool
8369     {
8370         \pgfpicture
8371         \pgfrememberpicturepositiononpagetrue
8372         \pgf@relevantforpicturesizefalse
8373         \pgfpathrectanglecorners
8374         { \@@_qpoint:n { 1 } }
8375         {
8376             \@@_qpoint:n
8377             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8378         }
8379         \pgfsetfillopacity { 0.75 }
8380         \pgfsetfillcolor { white }
8381         \pgfusepathqfill
8382         \endpgfpicture
8383     }
8384     \dim_zero_new:N \g_@@_tmpc_dim
8385     \dim_zero_new:N \g_@@_tmpd_dim
8386     \dim_zero_new:N \g_@@_tmpe_dim
8387     \int_step_inline:nn \c@iRow
8388     {
8389         \bool_if:NTF \l_@@_in_code_after_bool
8390         {
8391             \pgfpicture
8392             \pgfrememberpicturepositiononpagetrue
8393             \pgf@relevantforpicturesizefalse
8394         }
8395         { \begin { pgfpicture } }
8396         \@@_qpoint:n { row - ##1 }
8397         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8398         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8399         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8400         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8401         \bool_if:NTF \l_@@_in_code_after_bool
8402         { \endpgfpicture }
8403         { \end { pgfpicture } }
8404         \int_step_inline:nn \c@jCol
8405         {
8406             \hbox_set:Nn \l_tmpa_box
8407             {
8408                 \normalfont \Large \sffamily \bfseries
8409                 \bool_if:NTF \l_@@_in_code_after_bool
8410                 { \color { red } }
8411                 { \color { red ! 50 } }
8412                 ##1 - ####1
8413             }
8414             \bool_if:NTF \l_@@_in_code_after_bool
8415             {
8416                 \pgfpicture
8417                 \pgfrememberpicturepositiononpagetrue
8418                 \pgf@relevantforpicturesizefalse
8419             }
8420             { \begin { pgfpicture } }
8421             \@@_qpoint:n { col - ####1 }
8422             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8423             \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }

```

```

8424 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8425 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8426 \bool_if:NTF \l_@@_in_code_after_bool
8427 { \endpgfpicture }
8428 { \end { pgfpicture } }
8429 \fp_set:Nn \l_tmpa_fp
8430 {
8431   \fp_min:nn
8432   {
8433     \fp_min:nn
8434     { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8435     { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8436   }
8437   { 1.0 }
8438 }
8439 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8440 \pgfpicture
8441 \pgfrememberpicturepositiononpagetrue
8442 \pgf@relevantforpicturesizefalse
8443 \pgftransformshift
8444 {
8445   \pgfpoint
8446   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8447   { \dim_use:N \g_tmpa_dim }
8448 }
8449 \pgfnode
8450 { rectangle }
8451 { center }
8452 { \box_use:N \l_tmpa_box }
8453 { }
8454 { }
8455 \endpgfpicture
8456 }
8457 }
8458 }

```

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.

```

8459 \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.

```

8460 \bool_new:N \c_@@_footnote_bool
8461 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
8462 {
8463   The~key~'\l_keys_key_str'~is~unknown. \\
8464   That~key~will~be~ignored. \\
8465   For~a~list~of~the~available~keys,~type~H~<return>.
8466 }
8467 {
8468   The~available~keys~are~(in~alphabetic~order):~
8469   footnote,~
8470   footnotehyper,~
8471   messages-for-Overleaf,~

```

```

8472     no-test-for-array,~
8473     renew-dots,~and
8474     renew-matrix.
8475 }

8476 \keys_define:nn { NiceMatrix / Package }
8477 {
8478     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8479     renew-dots .value_forbidden:n = true ,
8480     renew-matrix .code:n = \@@_renew_matrix: ,
8481     renew-matrix .value_forbidden:n = true ,
8482     messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
8483     footnote .bool_set:N = \c_@@_footnote_bool ,
8484     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
8485     no-test-for-array .bool_set:N = \c_@@_no_test_for_array_bool ,
8486     no-test-for-array .default:n = true ,
8487     unknown .code:n = \@@_error:n { Unknown-key-for-package }
8488 }
8489 \ProcessKeysOptions { NiceMatrix / Package }

8490 \@@_msg_new:nn { footnote-with-footnotehyper-package }
8491 {
8492     You~can't~use~the~option~'footnote'~because~the~package~
8493     footnotehyper~has~already~been~loaded.~
8494     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8495     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8496     of~the~package~footnotehyper.\\
8497     The~package~footnote~won't~be~loaded.
8498 }

8499 \@@_msg_new:nn { footnotehyper-with-footnote-package }
8500 {
8501     You~can't~use~the~option~'footnotehyper'~because~the~package~
8502     footnote~has~already~been~loaded.~
8503     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8504     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8505     of~the~package~footnote.\\
8506     The~package~footnotehyper~won't~be~loaded.
8507 }

8508 \bool_if:NT \c_@@_footnote_bool
8509 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8510     \IfClassLoadedTF { beamer }
8511     { \bool_set_false:N \c_@@_footnote_bool }
8512     {
8513         \IfPackageLoadedTF { footnotehyper }
8514         { \@@_error:n { footnote-with-footnotehyper-package } }
8515         { \usepackage { footnote } }
8516     }
8517 }

8518 \bool_if:NT \c_@@_footnotehyper_bool
8519 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8520     \IfClassLoadedTF { beamer }
8521     { \bool_set_false:N \c_@@_footnote_bool }
8522     {
8523         \IfPackageLoadedTF { footnote }
8524         { \@@_error:n { footnotehyper-with-footnote-package } }

```

```

8525     { \usepackage { footnotehyper } }
8526   }
8527   \bool_set_true:N \c_@@_footnote_bool
8528 }

```

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.

```

8529 \bool_new:N \l_@@_underscore_loaded_bool
8530 \IfPackageLoadedTF { underscore }
8531 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8532 { }
8533 \hook_gput_code:nnn { begindocument } { . }
8534 {
8535   \bool_if:NF \l_@@_underscore_loaded_bool
8536   {
8537     \IfPackageLoadedTF { underscore }
8538     { \@@_error:n { underscore~after~nicematrix } }
8539     { }
8540   }
8541 }

```

38 Error messages of the package

```

8542 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
8543 { \str_const:Nn \c_@@_available_keys_str { } }
8544 {
8545   \str_const:Nn \c_@@_available_keys_str
8546   { For~a~list~of~the~available~keys,~type~H~<return>. }
8547 }
8548 \seq_new:N \g_@@_types_of_matrix_seq
8549 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8550 {
8551   NiceMatrix ,
8552   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8553 }
8554 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8555 { \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`.

```

8556 \cs_new_protected:Npn \@@_error_too_much_cols:
8557 {
8558   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8559   {
8560     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8561     { \@@_fatal:n { too-much-cols-for-matrix } }

```



```

8562     {
8563         \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8564         { \@@_fatal:n { too-much-cols-for-matrix } }
8565         {
8566             \bool_if:NF \l_@@_last_col_without_value_bool
8567             { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
8568         }
8569     }
8570 }
8571 {
8572     \IfPackageLoadedTF { tabularx }
8573     {
8574         \str_if_eq:VnTF \g_@@_name_env_str { NiceTabularX }
8575         {
8576             \int_compare:nNnTF \c@iRow = \c_zero_int
8577             { \@@_fatal:n { X-columns-with-tabularx } }
8578             {
8579                 \@@_fatal:nn { too-much-cols-for-array }
8580                 {
8581                     However,~this-message-may-be-erroneous:~
8582                     maybe-you-have-used~X-columns-while-'tabularx'~is~loaded,~
8583                     ~which-is-forbidden~(however,~it's~still~possible~to~use~
8584                     X-columns-in-{NiceTabularX}).
8585                 }
8586             }
8587         }
8588         { \@@_fatal:nn { too-much-cols-for-array } { } }
8589     }
8590     { \@@_fatal:nn { too-much-cols-for-array } { } }
8591 }
8592 }

```

The following command must *not* be protected since it's used in an error message.

```

8593 \cs_new:Npn \@@_message_hdotsfor:
8594 {
8595     \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8596     { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
8597 }
8598 \@@_msg_new:nn { negative-weight }
8599 {
8600     Negative-weight.\
8601     The-weight-of-the-'X'~columns-must-be-positive-and-you-have-used~
8602     the-value~'\int_use:N \l_@@_weight_int'~.\
8603     The-absolute-value-will-be-used.
8604 }
8605 \@@_msg_new:nn { last-col-not-used }
8606 {
8607     Column-not-used.\
8608     The-key-'last-col'~is~in-force-but-you-have-not-used-that-last-column~
8609     in-your~\@@_full_name_env:~.~However,~you-can-go-on.
8610 }
8611 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
8612 {
8613     Too-much-columns.\
8614     In-the-row~\int_eval:n { \c@iRow },~
8615     you-try-to-use-more-columns~
8616     than-allowed-by-your~\@@_full_name_env:~\@@_message_hdotsfor:\
8617     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
8618     (plus-the-exterior-columns).~This-error-is-fatal.
8619 }
8620 \@@_msg_new:nn { too-much-cols-for-matrix }
8621 {
8622     Too-much-columns.\

```

```

8623 In~the~row~\int_eval:n { \c@iRow },~
8624 you~try~to~use~more~columns~than~allowed~by~your~
8625 \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
8626 number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
8627 columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
8628 Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
8629 \token_to_str:N \setcounter\ to~change~that~value).~
8630 This~error~is~fatal.
8631 }

8632 \@@_msg_new:nn { too~much~cols~for~array }
8633 {
8634   Too~much~columns.\\
8635   In~the~row~\int_eval:n { \c@iRow },~
8636   ~you~try~to~use~more~columns~than~allowed~by~your~
8637   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
8638   \int_use:N \g_@@_static_num_of_col_int\
8639   ~(plus~the~potential~exterior~ones).~#1
8640   This~error~is~fatal.
8641 }

8642 \@@_msg_new:nn { X~columns~with~tabularx }
8643 {
8644   There~is~a~problem.\\
8645   You~have~probably~used~X~columns~in~your~environment~{\g_@@_name_env_str}.~
8646   That's~not~allowed~because~'tabularx'~is~loaded~(however,~you~can~use~X~columns~
8647   in~an~environment~{NiceTabularX}).\\
8648   This~error~is~fatal.
8649 }

8650 \@@_msg_new:nn { columns~not~used }
8651 {
8652   Columns~not~used.\\
8653   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8654   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
8655   The~columns~you~did~not~used~won't~be~created.\\
8656   We~won't~have~similar~error~till~the~end~of~the~document.
8657 }

8658 \@@_msg_new:nn { in~first~col }
8659 {
8660   Erroneous~use.\\
8661   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
8662   That~command~will~be~ignored.
8663 }

8664 \@@_msg_new:nn { in~last~col }
8665 {
8666   Erroneous~use.\\
8667   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
8668   That~command~will~be~ignored.
8669 }

8670 \@@_msg_new:nn { in~first~row }
8671 {
8672   Erroneous~use.\\
8673   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
8674   That~command~will~be~ignored.
8675 }

8676 \@@_msg_new:nn { in~last~row }
8677 {
8678   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
8679   That~command~will~be~ignored.
8680 }

8681 \@@_msg_new:nn { caption~outside~float }

```

```

8682 {
8683   Key~caption~forbidden.\\
8684   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
8685   environment.~This~key~will~be~ignored.
8686 }
8687 \@@_msg_new:nn { short-caption~without~caption }
8688 {
8689   You~should~not~use~the~key~'short-caption'~without~'caption'.~
8690   However,~your~'short-caption'~will~be~used~as~'caption'.
8691 }
8692 \@@_msg_new:nn { double-closing-delimiter }
8693 {
8694   Double~delimiter.\\
8695   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
8696   delimiter.~This~delimiter~will~be~ignored.
8697 }
8698 \@@_msg_new:nn { delimiter~after~opening }
8699 {
8700   Double~delimiter.\\
8701   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
8702   delimiter.~That~delimiter~will~be~ignored.
8703 }
8704 \@@_msg_new:nn { bad-option~for~line-style }
8705 {
8706   Bad~line~style.\\
8707   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
8708   is~'standard'.~That~key~will~be~ignored.
8709 }
8710 \@@_msg_new:nn { Identical~notes~in~caption }
8711 {
8712   Identical~tabular~notes.\\
8713   You~can't~put~several~notes~with~the~same~content~in~
8714   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
8715   If~you~go~on,~the~output~will~probably~be~erroneous.
8716 }
8717 \@@_msg_new:nn { tabularnote~below~the~tabular }
8718 {
8719   \token_to_str:N \tabularnote\ forbidden\\
8720   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8721   of~your~tabular~because~the~caption~will~be~composed~below~
8722   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8723   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
8724   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8725   no~similar~error~will~raised~in~this~document.
8726 }
8727 \@@_msg_new:nn { Unknown~key~for~rules }
8728 {
8729   Unknown~key.\\
8730   There~is~only~two~keys~available~here:~width~and~color.\\
8731   Your~key~'\l_keys_key_str'~will~be~ignored.
8732 }
8733 \@@_msg_new:nn { Unknown~key~for~rotate }
8734 {
8735   Unknown~key.\\
8736   The~only~key~available~here~is~'c'.\\
8737   Your~key~'\l_keys_key_str'~will~be~ignored.
8738 }
8739 \@@_msg_new:nnn { Unknown~key~for~custom~line }
8740 {
8741   Unknown~key.\\

```

```

8742 The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
8743 It~you~go~on,~you~will~probably~have~other~errors.~\
8744 \c_@@_available_keys_str
8745 }
8746 {
8747 The~available~keys~are~(in~alphabetic~order):~
8748 ccommand,~
8749 color,~
8750 command,~
8751 dotted,~
8752 letter,~
8753 multiplicity,~
8754 sep-color,~
8755 tikz,~and~total-width.
8756 }

8757 \@@_msg_new:nnn { Unknown~key~for~xdots }
8758 {
8759 Unknown~key.\
8760 The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\
8761 \c_@@_available_keys_str
8762 }
8763 {
8764 The~available~keys~are~(in~alphabetic~order):~
8765 'color',~
8766 'horizontal-labels',~
8767 'inter',~
8768 'line-style',~
8769 'radius',~
8770 'shorten',~
8771 'shorten-end'~and~'shorten-start'.
8772 }

8773 \@@_msg_new:nn { Unknown~key~for~rowcolors }
8774 {
8775 Unknown~key.\
8776 As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
8777 (and~you~try~to~use~'\l_keys_key_str')\
8778 That~key~will~be~ignored.
8779 }

8780 \@@_msg_new:nn { label~without~caption }
8781 {
8782 You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
8783 you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
8784 }

8785 \@@_msg_new:nn { W~warning }
8786 {
8787 Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
8788 (row~\int_use:N \c@iRow).
8789 }

8790 \@@_msg_new:nn { Construct~too~large }
8791 {
8792 Construct~too~large.\
8793 Your~command~\token_to_str:N #1
8794 can't~be~drawn~because~your~matrix~is~too~small.\
8795 That~command~will~be~ignored.
8796 }

8797 \@@_msg_new:nn { underscore~after~nicematrix }
8798 {
8799 Problem~with~'underscore'.\
8800 The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
8801 You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\
8802 '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.

```

```

8803     }
8804 \@@_msg_new:nn { ampersand~in~light-syntax }
8805 {
8806     Ampersand~forbidden.\\
8807     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
8808     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
8809 }
8810 \@@_msg_new:nn { double-backslash~in~light-syntax }
8811 {
8812     Double~backslash~forbidden.\\
8813     You~can't~use~\token_to_str:N
8814     \\~to~separate~rows~because~the~key~'light-syntax'~
8815     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
8816     (set~by~the~key~'end-of-row').~This~error~is~fatal.
8817 }
8818 \@@_msg_new:nn { hlines~with~color }
8819 {
8820     Incompatible~keys.\\
8821     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
8822     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
8823     Maybe~it~will~possible~in~future~version.\\
8824     Your~key~will~be~discarded.
8825 }
8826 \@@_msg_new:nn { bad~value~for~baseline }
8827 {
8828     Bad~value~for~baseline.\\
8829     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
8830     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
8831     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
8832     the~form~'line-i'.\\
8833     A~value~of~1~will~be~used.
8834 }
8835 \@@_msg_new:nn { ragged2e~not~loaded }
8836 {
8837     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
8838     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
8839     \l_keys_key_str'~will~be~used~instead.
8840 }
8841 \@@_msg_new:nn { Invalid~name }
8842 {
8843     Invalid~name.\\
8844     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
8845     \SubMatrix\ of~your~\@@_full_name_env:.\\
8846     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
8847     This~key~will~be~ignored.
8848 }
8849 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
8850 {
8851     Wrong~line.\\
8852     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
8853     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
8854     number~is~not~valid.~It~will~be~ignored.
8855 }
8856 \@@_msg_new:nn { Impossible~delimiter }
8857 {
8858     Impossible~delimiter.\\
8859     It's~impossible~to~draw~the~#1~delimiter~of~your~
8860     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
8861     in~that~column.
8862     \bool_if:NT \l_@@_submatrix_slim_bool

```

```

8863     { ~Maybe-you-should-try-without-the-key-'slim'. } \\
8864     This~\token_to_str:N \SubMatrix\ will-be-ignored.
8865 }

8866 \@@_msg_new:nn { width-without-X-columns }
8867 {
8868     You-have-used-the-key-'width'-but-you-have-put-no-'X'-column.~
8869     That-key-will-be-ignored.
8870 }

8871 \@@_msg_new:nn { key-multiplicity-with-dotted }
8872 {
8873     Incompatible-keys. \\
8874     You-have-used-the-key-'multiplicity'~with~the-key-'dotted'~
8875     in~a-'custom-line'.~They-are-incompatible. \\
8876     The-key-'multiplicity'~will-be-discarded.
8877 }

8878 \@@_msg_new:nn { empty-environment }
8879 {
8880     Empty-environment.\\
8881     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
8882 }

8883 \@@_msg_new:nn { No-letter-and-no-command }
8884 {
8885     Erroneous-use.\\
8886     Your-use-of-'custom-line'~is-no-op~since~you~don't~have~used~the~
8887     key-'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys-'command'~or~
8888     '~'ccommand'~(to~draw~horizontal~rules).\\
8889     However,~you~can~go~on.
8890 }

8891 \@@_msg_new:nn { Forbidden-letter }
8892 {
8893     Forbidden-letter.\\
8894     You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
8895     It~will~be~ignored.
8896 }

8897 \@@_msg_new:nn { Several-letters }
8898 {
8899     Wrong-name.\\
8900     You~must~use~only~one~letter~as~value~for~the~key-'letter'~(and~you~
8901     have~used~'\l_@@_letter_str').\\
8902     It~will~be~ignored.
8903 }

8904 \@@_msg_new:nn { Delimiter-with-small }
8905 {
8906     Delimiter~forbidden.\\
8907     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
8908     because~the~key-'small'~is~in~force.\\
8909     This~error~is~fatal.
8910 }

8911 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
8912 {
8913     Unknown~cell.\\
8914     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
8915     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
8916     can't~be~executed~because~a~cell~doesn't~exist.\\
8917     This~command~\token_to_str:N \line\ will-be-ignored.
8918 }

8919 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
8920 {
8921     Duplicate~name.\\
8922     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\

```

```

8923   in~this~\@@_full_name_env:.\
8924   This~key~will~be~ignored.\\
8925   \bool_if:NF \c_@@_messages_for_Overleaf_bool
8926   { For~a~list~of~the~names~already~used,~type~H~<return>. }
8927 }
8928 {
8929   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
8930   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
8931 }
8932 \@@_msg_new:nn { r-or-l-with-preamble }
8933 {
8934   Erroneous~use.\\
8935   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
8936   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
8937   your~\@@_full_name_env:.\
8938   This~key~will~be~ignored.
8939 }
8940 \@@_msg_new:nn { Hdotsfor~in~col~0 }
8941 {
8942   Erroneous~use.\\
8943   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
8944   the~array.~This~error~is~fatal.
8945 }
8946 \@@_msg_new:nn { bad-corner }
8947 {
8948   Bad~corner.\\
8949   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
8950   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
8951   This~specification~of~corner~will~be~ignored.
8952 }
8953 \@@_msg_new:nn { bad-border }
8954 {
8955   Bad~border.\\
8956   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
8957   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
8958   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
8959   also~use~the~key~'tikz'
8960   \IfPackageLoadedTF { tikz }
8961   { }
8962   {~if~you~load~the~LaTeX~package~'tikz'}).\\
8963   This~specification~of~border~will~be~ignored.
8964 }
8965 \@@_msg_new:nn { tikz~key~without~tikz }
8966 {
8967   Tikz~not~loaded.\\
8968   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
8969   \Block'~because~you~have~not~loaded~tikz.~
8970   This~key~will~be~ignored.
8971 }
8972 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
8973 {
8974   Erroneous~use.\\
8975   In~the~\@@_full_name_env:,~you~must~use~the~key~
8976   'last-col'~without~value.\\
8977   However,~you~can~go~on~for~this~time~
8978   (the~value~'\l_keys_value_tl'~will~be~ignored).
8979 }
8980 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
8981 {
8982   Erroneous~use.\\
8983   In~\NiceMatrixoptions,~you~must~use~the~key~

```

```

8984     'last-col'~without~value.\\
8985     However,~you~can~go~on~for~this~time~
8986     (the~value~'\l_keys_value_tl'~will~be~ignored).
8987 }

8988 \@@_msg_new:nn { Block~too~large-1 }
8989 {
8990     Block~too~large.\\
8991     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
8992     too~small~for~that~block. \\
8993 }

8994 \@@_msg_new:nn { Block~too~large-2 }
8995 {
8996     Block~too~large.\\
8997     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8998     \g_@@_static_num_of_col_int\
8999     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9000     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9001     (&)~at~the~end~of~the~first~row~of~your~
9002     \@@_full_name_env:.\\
9003     This~block~and~maybe~others~will~be~ignored.
9004 }

9005 \@@_msg_new:nn { unknown~column~type }
9006 {
9007     Bad~column~type.\\
9008     The~column~type~'#1'~in~your~\@@_full_name_env:\
9009     is~unknown. \\
9010     This~error~is~fatal.
9011 }

9012 \@@_msg_new:nn { unknown~column~type-S }
9013 {
9014     Bad~column~type.\\
9015     The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9016     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9017     load~that~package. \\
9018     This~error~is~fatal.
9019 }

9020 \@@_msg_new:nn { tabularnote~forbidden }
9021 {
9022     Forbidden~command.\\
9023     You~can't~use~the~command~\token_to_str:N\tabularnote\
9024     ~here.~This~command~is~available~only~in~
9025     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9026     the~argument~of~a~command~\token_to_str:N \caption\ included~
9027     in~an~environment~{table}.\\
9028     This~command~will~be~ignored.
9029 }

9030 \@@_msg_new:nn { borders~forbidden }
9031 {
9032     Forbidden~key.\\
9033     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9034     because~the~option~'rounded~corners'~
9035     is~in~force~with~a~non~zero~value.\\
9036     This~key~will~be~ignored.
9037 }

9038 \@@_msg_new:nn { bottomrule~without~booktabs }
9039 {
9040     booktabs~not~loaded.\\
9041     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9042     loaded~'booktabs'.\\
9043     This~key~will~be~ignored.
9044 }

```



```

9045 \@@_msg_new:nn { enumitem~not~loaded }
9046 {
9047     enumitem~not~loaded.\\
9048     You~can't~use~the~command~\token_to_str:N\tabularnote\
9049     ~because~you~haven't~loaded~'enumitem'.\\
9050     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9051     ignored~in~the~document.
9052 }
9053 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9054 {
9055     Tikz~not~loaded.\\
9056     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9057     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9058     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9059     use~that~custom~line.
9060 }
9061 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9062 {
9063     Tikz~not~loaded.\\
9064     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9065     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9066     That~key~will~be~ignored.
9067 }
9068 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9069 {
9070     Erroneous~use.\\
9071     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9072     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9073     The~key~'color'~will~be~discarded.
9074 }
9075 \@@_msg_new:nn { Wrong~last~row }
9076 {
9077     Wrong~number.\\
9078     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9079     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9080     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9081     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9082     without~value~(more~compilations~might~be~necessary).
9083 }
9084 \@@_msg_new:nn { Yet~in~env }
9085 {
9086     Nested~environments.\\
9087     Environments~of~nicematrix~can't~be~nested.\\
9088     This~error~is~fatal.
9089 }
9090 \@@_msg_new:nn { Outside~math~mode }
9091 {
9092     Outside~math~mode.\\
9093     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9094     (and~not~in~\token_to_str:N \vcenter).\\
9095     This~error~is~fatal.
9096 }
9097 \@@_msg_new:nn { One~letter~allowed }
9098 {
9099     Bad~name.\\
9100     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9101     It~will~be~ignored.
9102 }
9103 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9104 {

```

```

9105 Environment~{TabularNote}~forbidden.\\
9106 You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9107 but~*before*~the~\token_to_str:N \CodeAfter.\\
9108 This~environment~{TabularNote}~will~be~ignored.
9109 }
9110 \@@_msg_new:nn { varwidth~not~loaded }
9111 {
9112   varwidth~not~loaded.\\
9113   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9114   loaded.\\
9115   Your~column~will~behave~like~'p'.
9116 }
9117 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
9118 {
9119   Unknown~key.\\
9120   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9121   \c_@@_available_keys_str
9122 }
9123 {
9124   The~available~keys~are~(in~alphabetic~order):~
9125   color,~
9126   dotted,~
9127   multiplicity,~
9128   sep-color,~
9129   tikz,~and~total~width.
9130 }
9131
9132 \@@_msg_new:nnn { Unknown~key~for~Block }
9133 {
9134   Unknown~key.\\
9135   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9136   \Block.\\ It~will~be~ignored. \\
9137   \c_@@_available_keys_str
9138 }
9139 {
9140   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9141   hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,~
9142   t,~T,~tikz,~transparent~and~vlines.
9143 }
9144 \@@_msg_new:nn { Version~of~siunitx~too~old }
9145 {
9146   siunitx~too~old.\\
9147   You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9148   is~too~old.~You~need~at~least~v3.0.38~and~your~log~file~says:~"siunitx,~
9149   \use:c { ver @ siunitx.sty }". \\
9150   This~error~is~fatal.
9151 }
9152 \@@_msg_new:nnn { Unknown~key~for~Brace }
9153 {
9154   Unknown~key.\\
9155   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9156   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9157   It~will~be~ignored. \\
9158   \c_@@_available_keys_str
9159 }
9160 {
9161   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9162   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9163   right~shorten)~and~yshift.
9164 }
9165 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }

```

```

9166 {
9167     Unknown~key.\\
9168     The~key~'\l_keys_key_str'~is-unknown.\\
9169     It~will~be~ignored. \\
9170     \c_@@_available_keys_str
9171 }
9172 {
9173     The~available~keys~are~(in~alphabetic~order):~
9174     delimiters/color,~
9175     rules~(with~the~subkeys~'color'~and~'width'),~
9176     sub-matrix~(several~subkeys)~
9177     and~xdots~(several~subkeys).~
9178     The~latter~is~for~the~command~\token_to_str:N \line.
9179 }
9180 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9181 {
9182     Unknown~key.\\
9183     The~key~'\l_keys_key_str'~is-unknown.\\
9184     It~will~be~ignored. \\
9185     \c_@@_available_keys_str
9186 }
9187 {
9188     The~available~keys~are~(in~alphabetic~order):~
9189     create-cell-nodes,~
9190     delimiters/color~and~
9191     sub-matrix~(several~subkeys).
9192 }
9193 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9194 {
9195     Unknown~key.\\
9196     The~key~'\l_keys_key_str'~is-unknown.\\
9197     That~key~will~be~ignored. \\
9198     \c_@@_available_keys_str
9199 }
9200 {
9201     The~available~keys~are~(in~alphabetic~order):~
9202     'delimiters/color',~
9203     'extra-height',~
9204     'hlines',~
9205     'hvlines',~
9206     'left-xshift',~
9207     'name',~
9208     'right-xshift',~
9209     'rules'~(with~the~subkeys~'color'~and~'width'),~
9210     'slim',~
9211     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9212     and~'right-xshift').\\
9213 }
9214 \@@_msg_new:nnn { Unknown~key~for~notes }
9215 {
9216     Unknown~key.\\
9217     The~key~'\l_keys_key_str'~is-unknown.\\
9218     That~key~will~be~ignored. \\
9219     \c_@@_available_keys_str
9220 }
9221 {
9222     The~available~keys~are~(in~alphabetic~order):~
9223     bottomrule,~
9224     code-after,~
9225     code-before,~
9226     detect-duplicates,~
9227     enumitem-keys,~
9228     enumitem-keys-para,~

```

```

9229     para,~
9230     label-in-list,~
9231     label-in-tabular-and~
9232     style.
9233 }
9234 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9235 {
9236     Unknown~key.\\
9237     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9238     \token_to_str:N \RowStyle. \\
9239     That~key~will~be~ignored. \\
9240     \c_@@_available_keys_str
9241 }
9242 {
9243     The~available~keys~are~(in~alphabetic~order):~
9244     'bold',~
9245     'cell-space-top-limit',~
9246     'cell-space-bottom-limit',~
9247     'cell-space-limits',~
9248     'color',~
9249     'nb-rows'~and~
9250     'rowcolor'.
9251 }
9252 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9253 {
9254     Unknown~key.\\
9255     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9256     \token_to_str:N \NiceMatrixOptions. \\
9257     That~key~will~be~ignored. \\
9258     \c_@@_available_keys_str
9259 }
9260 {
9261     The~available~keys~are~(in~alphabetic~order):~
9262     allow-duplicate-names,~
9263     caption-above,~
9264     cell-space-bottom-limit,~
9265     cell-space-limits,~
9266     cell-space-top-limit,~
9267     code-for-first-col,~
9268     code-for-first-row,~
9269     code-for-last-col,~
9270     code-for-last-row,~
9271     corners,~
9272     custom-key,~
9273     create-extra-nodes,~
9274     create-medium-nodes,~
9275     create-large-nodes,~
9276     delimiters~(several-subkeys),~
9277     end-of-row,~
9278     first-col,~
9279     first-row,~
9280     hlines,~
9281     hvlines,~
9282     hvlines-except-borders,~
9283     last-col,~
9284     last-row,~
9285     left-margin,~
9286     light-syntax,~
9287     matrix/columns-type,~
9288     notes~(several-subkeys),~
9289     nullify-dots,~
9290     pgf-node-code,~
9291     renew-dots,~

```

```

9292   renew-matrix,~
9293   respect-arraystretch,~
9294   rounded-corners,~
9295   right-margin,~
9296   rules~(with~the~subkeys~'color'~and~'width'),~
9297   small,~
9298   sub-matrix~(several~subkeys),~
9299   vlines,~
9300   xdots~(several~subkeys).
9301 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

9302 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9303 {
9304   Unknown~key.\\
9305   The~key~'\l_keys_str'~is~unknown~for~the~environment~
9306   \{NiceArray\}. \\
9307   That~key~will~be~ignored. \\
9308   \c_@@_available_keys_str
9309 }
9310 {
9311   The~available~keys~are~(in~alphabetic~order):~
9312   b,~
9313   baseline,~
9314   c,~
9315   cell-space-bottom-limit,~
9316   cell-space-limits,~
9317   cell-space-top-limit,~
9318   code-after,~
9319   code-for-first-col,~
9320   code-for-first-row,~
9321   code-for-last-col,~
9322   code-for-last-row,~
9323   color-inside,~
9324   columns-width,~
9325   corners,~
9326   create-extra-nodes,~
9327   create-medium-nodes,~
9328   create-large-nodes,~
9329   extra-left-margin,~
9330   extra-right-margin,~
9331   first-col,~
9332   first-row,~
9333   hlines,~
9334   hvlines,~
9335   hvlines-except-borders,~
9336   last-col,~
9337   last-row,~
9338   left-margin,~
9339   light-syntax,~
9340   name,~
9341   nullify-dots,~
9342   pgf-node-code,~
9343   renew-dots,~
9344   respect-arraystretch,~
9345   right-margin,~
9346   rounded-corners,~
9347   rules~(with~the~subkeys~'color'~and~'width'),~
9348   small,~
9349   t,~
9350   vlines,~
9351   xdots/color,~
9352   xdots/shorten-start,~

```

```

9353     xdots/shorten-end,~
9354     xdots/shorten~and~
9355     xdots/line-style.
9356 }

```

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).

```

9357 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9358 {
9359   Unknown~key.\\
9360   The~key~'\l_keys_key_str'~is~unknown~for~the~
9361   \@@_full_name_env:. \\
9362   That~key~will~be~ignored. \\
9363   \c_@@_available_keys_str
9364 }
9365 {
9366   The~available~keys~are~(in~alphabetic~order):~
9367   b,~
9368   baseline,~
9369   c,~
9370   cell-space-bottom-limit,~
9371   cell-space-limits,~
9372   cell-space-top-limit,~
9373   code-after,~
9374   code-for-first-col,~
9375   code-for-first-row,~
9376   code-for-last-col,~
9377   code-for-last-row,~
9378   color-inside,~
9379   columns-type,~
9380   columns-width,~
9381   corners,~
9382   create-extra-nodes,~
9383   create-medium-nodes,~
9384   create-large-nodes,~
9385   extra-left-margin,~
9386   extra-right-margin,~
9387   first-col,~
9388   first-row,~
9389   hlines,~
9390   hvlines,~
9391   hvlines-except-borders,~
9392   l,~
9393   last-col,~
9394   last-row,~
9395   left-margin,~
9396   light-syntax,~
9397   name,~
9398   nullify-dots,~
9399   pgf-node-code,~
9400   r,~
9401   renew-dots,~
9402   respect-arraystretch,~
9403   right-margin,~
9404   rounded-corners,~
9405   rules~(with~the~subkeys~'color'~and~'width'),~
9406   small,~
9407   t,~
9408   vlines,~
9409   xdots/color,~
9410   xdots/shorten-start,~
9411   xdots/shorten-end,~
9412   xdots/shorten~and~
9413   xdots/line-style.

```

```

9414     }
9415 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9416 {
9417     Unknown~key.\\
9418     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9419     \{NiceTabular\}. \\
9420     That~key~will~be~ignored. \\
9421     \c_@@_available_keys_str
9422 }
9423 {
9424     The~available~keys~are~(in~alphabetic~order):~
9425     b,~
9426     baseline,~
9427     c,~
9428     caption,~
9429     cell-space-bottom-limit,~
9430     cell-space-limits,~
9431     cell-space-top-limit,~
9432     code-after,~
9433     code-for-first-col,~
9434     code-for-first-row,~
9435     code-for-last-col,~
9436     code-for-last-row,~
9437     color-inside,~
9438     columns-width,~
9439     corners,~
9440     custom-line,~
9441     create-extra-nodes,~
9442     create-medium-nodes,~
9443     create-large-nodes,~
9444     extra-left-margin,~
9445     extra-right-margin,~
9446     first-col,~
9447     first-row,~
9448     hlines,~
9449     hvlines,~
9450     hvlines-except-borders,~
9451     label,~
9452     last-col,~
9453     last-row,~
9454     left-margin,~
9455     light-syntax,~
9456     name,~
9457     notes~(several~subkeys),~
9458     nullify-dots,~
9459     pgf-node-code,~
9460     renew-dots,~
9461     respect-arraystretch,~
9462     right-margin,~
9463     rounded-corners,~
9464     rules~(with~the~subkeys~'color'~and~'width'),~
9465     short-caption,~
9466     t,~
9467     tabularnote,~
9468     vlines,~
9469     xdots/color,~
9470     xdots/shorten-start,~
9471     xdots/shorten-end,~
9472     xdots/shorten-and~
9473     xdots/line-style.
9474 }
9475 \@@_msg_new:nnn { Duplicate-name }
9476 {

```

```

9477 Duplicate~name.\\
9478 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9479 the~same~environment~name~twice.~You~can~go~on,~but,~
9480 maybe,~you~will~have~incorrect~results~especially~
9481 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9482 message~again,~use~the~key~'allow-duplicate-names'~in~
9483 '\token_to_str:N \NiceMatrixOptions'.\\
9484 \bool_if:NF \c_@@_messages_for_Overleaf_bool
9485   { For~a~list~of~the~names~already~used,~type~H~<return>. }
9486 }
9487 {
9488   The~names~already~defined~in~this~document~are:~
9489   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9490 }
9491 \@@_msg_new:nn { Option~auto~for~columns~width }
9492 {
9493   Erroneous~use.\\
9494   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9495   That~key~will~be~ignored.
9496 }

```


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>	18
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>	87
15	After the construction of the array	89
16	We draw the dotted lines	95
17	The actual instructions for drawing the dotted lines with <code>Tikz</code>	107
18	User commands available in the new environments	111
19	The command <code>\line</code> accessible in code-after	117
20	The command <code>\RowStyle</code>	119
21	Colors of cells, rows and columns	121
22	The vertical and horizontal rules	130
23	The key corners	145
24	The environment <code>{NiceMatrixBlock}</code>	147
25	The extra nodes	148
26	The blocks	153
27	How to draw the dotted lines transparently	171
28	Automatic arrays	171
29	The redefinition of the command <code>\dotfill</code>	172
30	The command <code>\diagbox</code>	173

31	The keyword <code>\CodeAfter</code>	174
32	The delimiters in the preamble	175
33	The command <code>\SubMatrix</code>	176
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	185
35	The command <code>\ShowCellNames</code>	188
36	We process the options at package loading	190
37	About the package underscore	192
38	Error messages of the package	192