

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

June 30, 2023

Abstract

This document is the documented code of the LaTeX package `nicematrix`. It is *not* its user's guide. The guide of utilisation is the document `nicematrix.pdf` (with a French traduction: `nicematrix-french.pdf`).

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

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

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

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

1 Declaration of the package and packages loaded

The prefix `nicematrix` has been registered for this package.

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

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

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

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

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

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

We load some packages.

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

*This document corresponds to the version 6.20a of `nicematrix`, at the date of 2023/06/30.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

2 Security test

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

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

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

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

```

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

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

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

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

```

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

```

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

```

3 Technical definitions

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

203     \skip_horizontal:N \c_zero_dim
204 }

```

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

```

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

```

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

```

209 \cs_set:Npn \@@_cline

```

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

```

210 { \@@_cline_i:en \l_@@_first_col_int }

```

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

```

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

```

¹See question 99041 on TeX StackExchange.

```

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

```

Now, #1 is the number of the current column and we have to draw a line from the column #2 to the column #3 (both included).

```

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

```

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

```

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

```

The following command is a small shortcut.

```

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

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

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

```

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

```

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

```

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

```

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

```

```

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

264 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

```

The column S of siunitx

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

```

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

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

```

4 Parameters

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

```

298 \int_new:N \g_@@_env_int

```

The following command is only a syntactic shortcut. It must *not* be protected (it will be used in names of PGF nodes).

```

299 \cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }

```

The command `\NiceMatrixLastEnv` is not used by the package `nicematrix`. It’s only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it’s meant to be used after the environment in order to refer to that environment

— and its nodes — without having to give it a name). This command *must* be expandable since it will be used in pgf nodes.

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

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

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

The following counter will count the environments {NiceMatrixBlock}.

```
304 \int_new:N \g_@@_NiceMatrixBlock_int
```

It's possible to put tabular notes (with \tabularnote) in the caption if that caption is composed *above* the tabular. In such case, we will count in \g_@@_notes_caption_int the number of uses of the command \tabularnote *without optional argument* in that caption.

```
305 \int_new:N \g_@@_notes_caption_int
```

The dimension \l_@@_columns_width_dim will be used when the options specify that all the columns must have the same width (but, if the key columns-width is used with the special value auto, the boolean l_@@_auto_columns_width_bool also will be raised).

```
306 \dim_new:N \l_@@_columns_width_dim
```

The dimension \l_@@_col_width_dim will be available in each cell which belongs to a column of fixed width: w{...}{...}, W{...}{...}, p{ }, m{ }, b{ } but also X (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands \Block. A non positive value means that the column has no fixed width (it's a column of type c, r, l, etc.).

```
307 \dim_new:N \l_@@_col_width_dim
308 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

The following counters will be used to count the numbers of rows and columns of the array.

```
309 \int_new:N \g_@@_row_total_int
310 \int_new:N \g_@@_col_total_int
```

The following parameter will be used by \@@_create_row_node: to avoid to create the same row-node twice (at the end of the array).

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

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

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

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

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

When there is a mono-column block (created by the command \Block), we want to take into account the width of that block for the width of the column. That's why we compute the width of that block in the \g_@@_blocks_wd_dim and, after the construction of the box \l_@@_cell_box, we change the width of that box to take into account the length \g_@@_blocks_wd_dim.

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

Idem for the mono-row blocks.

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

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

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

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

```
319 \dim_new:N \l_@@_width_dim
```

The sequence `\g_@@_names_seq` will be the list of all the names of environments used (via the option `name`) in the document: two environments must not have the same name. However, it's possible to use the option `allow-duplicate-names`.

```
320 \seq_new:N \g_@@_names_seq
```

We want to know whether we are in an environment of `nicematrix` because we will raise an error if the user tries to use nested environments.

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

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

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

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

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

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

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

```
325 \bool_new:N \l_@@_NiceTabular_bool
```

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

```
326 \dim_new:N \l_@@_tabular_width_dim
```

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

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

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

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

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

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

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

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

```
331 \bool_new:N \g_@@_caption_finished_bool
```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`).

```
332 \tl_new:N \g_@@_aux_tl
```

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`. However, it does *not* contain the value provided by the final user. Indeed, a transformation is done in order to have a preamble (for the package `array`) which is `nicematrix`-aware. That transformation is done with the command `\@@_set_preamble:Nn`.

```

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

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

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

```

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

The list of the columns where vertical lines in sub-matrices (`vlism`) must be drawn. Of course, the actual value of this sequence will be known after the analyse of the preamble of the array.

```

343 \seq_new:N \g_@@_cols_vlism_seq
```

The following colors will be used to memorize the color of the potential “first col” and the potential “first row”.

```

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

The following string is the name of the current environment or the current command of `nicematrix` (despite its name which contains `env`).

```

346 \str_new:N \g_@@_name_env_str
```

The following string will contain the word *command* or *environment* whether we are in a command of `nicematrix` or in an environment of `nicematrix`. The default value is *environment*.

```

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

The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains `env`). This command must *not* be protected since it will be used in error messages and we have to use `\str_if_eq:VnTF` and not `\tl_if_eq:NnTF` because we need to be fully expandable).

```

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

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

```

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

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

```

357 \tl_new:N \l_@@_code_tl
```

For the key `pgf-node-code`. That code will be used when the nodes of the cells (that is to say the nodes of the form `i-j`) will be created.

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

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

```
359 \tl_new:N \g_@@_pre_code_after_tl
```

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

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

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

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

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

```
363 \int_new:N \l_@@_old_jCol_int
```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

The following sequence will contain the names (without backslash) of the commands created by `custom-line` by the key `command` or `ccommand` (commands used by the final user in order to draw horizontal rules).

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

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

```
365 \tl_new:N \l_@@_rules_color_tl
```

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as an optional argument between square brackets. The default value, of course, is 1.

```
366 \int_new:N \g_@@_total_X_weight_int
```

If there is at least one X-column in the preamble of the array, the following flag will be raised via the `aux` file. The length `\l_@@_x_columns_dim` will be the width of X-columns of weight 1 (the width of a column of weight n will be that dimension multiplied by n). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

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

```
368 \dim_new:N \l_@@_X_columns_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
369 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the `col` nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of `col` nodes).

```
370 \bool_new:N \g_@@_row_of_col_done_bool
```

It's possible to use the command `\NotEmpty` to specify explicitly that a cell must be considered as non empty by `nicematrix` (the Tikz nodes are constructed only in the non empty cells).

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

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

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

```
372 \tl_new:N \l_@@_code_before_tl
373 \bool_new:N \l_@@_code_before_bool
```

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

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

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

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

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

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

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

```
381 \bool_new:N \g_@@_empty_cell_bool
```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

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

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces:

`{imin}{jmin}{imax}{jmax}{options}{contents}`.

The variable is global because it will be modified in the cells of the array.

```
384 \seq_new:N \g_@@_blocks_seq
```

We also manage a sequence of the *positions* of the blocks. In that sequence, each block is represented by only five components: `{imin}{jmin}{imax}{jmax}{ name}`. A block with the key `hvlines` won’t appear in that sequence (otherwise, the lines in that block would not be drawn!).

```
385 \seq_new:N \g_@@_pos_of_blocks_seq
```

In fact, this sequence will also contain the positions of the cells with a `\diagbox`. The sequence `\g_@@_pos_of_blocks_seq` will be used when we will draw the rules (which respect the blocks).

We will also manage a sequence for the positions of the dotted lines. These dotted lines are created in the array by `\Cdots`, `\Vdots`, `\Ddots`, etc. However, their positions, that is to say, their extremities, will be determined only after the construction of the array. In this sequence, each item contains five components: `{imin}{jmin}{imax}{jmax}{ name}`.

```
386 \seq_new:N \g_@@_pos_of_xdots_seq
```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won't be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That's why we keep the information of all that stroken blocks in the following sequence.

```
387 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following sequence.

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

The list of the names of the potential `\SubMatrix` in the `\CodeAfter` of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given `\SubMatrix`).

```
389 \seq_new:N \g_@@_submatrix_names_seq
```

The following flag will be raised if the key `width` is used in an environment `\NiceTabular` (not in a command `\NiceMatrixOptions`). You use it to raise an error when this key is used while no column `X` is used.

```
390 \bool_new:N \l_@@_width_used_bool
```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with $n > 1$ is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of n) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Variables for the exterior rows and columns

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

- **First row**

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

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

- **First column**

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

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

- **Last row**

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

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

If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.²

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

For the potential “last column”, we use an integer. A value of `-2` means that there is no last column. A value of `-1` means that we are in an environment without preamble (e.g. `{bNiceMatrix}`) and there is a last column but we don’t know its value because the user has used the option `last-col` without value. A value of 0 means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`): in this case, the key was necessary without argument.

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

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

²We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.


```

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

```

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

```

436 \bool_new:N \g_@@_last_col_found_bool

```

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

Some utilities

```

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

```

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers.

```

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

```

5 The command `\tabularnote`

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

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

- The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.³
- During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each 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.

```

461 \newcounter { tabularnote }
462 \seq_new:N \g_@@_notes_seq
463 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

464 \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.

```

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

```

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

```

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

```

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

```

473 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }

```

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

```

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

```

We define `\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`.

```

475 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }

```

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

```

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

```

The type of list `tabularnotes` will be used to format the tabular notes at the end of the array in the general case and `tabularnotes*` will be used if the key `para` is in force.

```

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

```

One must remind that we have allowed a `\tabular` in the caption and that caption may also be found in the list of tables (`\listoftables`). We want the command `\tabularnote` be no-op during the composition of that list. That's why we program `\tabularnote` to be no-op excepted in a floating environment or in an environment of `nicematrix`.

```

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

```

For the version in normal conditions, that is to say not in the `caption`. `#1` is the optional argument of `\tabularnote` (maybe equal to the special marker `\c_novalue_tl`) and `#2` is the mandatory argument of `\tabularnote`.

```

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

```

You have to see whether the argument of `\tabularnote` has yet been used as argument of another `\tabularnote` in the same tabular. In that case, there will be only one note (for both commands `\tabularnote`) at the end of the tabular. We search the argument of our command `\tabularnote` in `\g_@@_notes_seq`. The position in the sequence will be stored in `\l_tmpa_int` (0 if the text is not in the sequence yet).

```

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

```

We recall that each component of `\g_@@_notes_seq` is a kind of couple of the form

`{label}{text of the tabularnote}`.

If the user have used `\tabularnote` without the optional argument, the *label* will be the special marker `\c_novalue_tl`.

```

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

```

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

```

560   \hbox_set:Nn \l_tmpa_box
561   {

```

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

```

562   \@@_notes_label_in_tabular:n
563   {

```

```

564         \seq_use:Nnnn
565         \l_@@_notes_labels_seq { , } { , } { , }
566     }
567 }

```

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

```

568     \int_gsub:Nn \c@tabularnote { 1 }
569     \int_set_eq:NN \l_tmpa_int \c@tabularnote
570     \refstepcounter { tabularnote }
571     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
572     { \int_gincr:N \c@tabularnote }
573     \seq_clear:N \l_@@_notes_labels_seq
574     \bool_lazy_or:nnTF
575     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { c } }
576     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { r } }
577     {
578         \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).

```

579         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
580     }
581     { \box_use:N \l_tmpa_box }
582 }
583 }

```

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`.

```

584 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
585 {
586     \bool_if:NTF \g_@@_caption_finished_bool
587     {
588         \int_compare:nNnT
589         \c@tabularnote = \g_@@_notes_caption_int
590         { \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`!

```

591         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
592         { \@@_error:n { Identical~notes~in~caption } }
593     }
594 }

```

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

```

595     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
596     {

```

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.

```

597         \bool_gset_true:N \g_@@_caption_finished_bool
598         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
599         \int_gzero:N \c@tabularnote
600     }
601     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
602 }

```

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!

```

603 \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
604 \seq_put_right:Nx \l_@@_notes_labels_seq
605 {
606   \tl_if_novalue:nTF { #1 }
607     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
608     { #1 }
609 }
610 \peek_meaning:NF \tabularnote
611 {
612   \@@_notes_label_in_tabular:n
613   { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
614   \seq_clear:N \l_@@_notes_labels_seq
615 }
616 }
617 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
618 { \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.

```

619 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
620 {
621   \begin { pgfscope }
622   \pgfset
623   {
624     % outer~sep = \c_zero_dim ,
625     inner~sep = \c_zero_dim ,
626     minimum~size = \c_zero_dim
627   }
628   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
629   \pgfnode
630   { rectangle }
631   { center }
632   {
633     \vbox_to_ht:nn
634     { \dim_abs:n { #5 - #3 } }
635     {
636       \vfill
637       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
638     }
639   }
640   { #1 }
641   { }
642   \end { pgfscope }
643 }

```

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.

```

644 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
645 {
646   \begin { pgfscope }
647   \pgfset
648   {
649     % outer~sep = \c_zero_dim ,

```

```

650         inner~sep = \c_zero_dim ,
651         minimum~size = \c_zero_dim
652     }
653     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
654     \pgfpointdiff { #3 } { #2 }
655     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
656     \pgfnode
657     { rectangle }
658     { center }
659     {
660         \vbox_to_ht:nn
661         { \dim_abs:n \l_tmpb_dim }
662         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
663     }
664     { #1 }
665     { }
666     \end { pgfscope }
667 }

```

7 The options

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

```

668 \tl_new:N \l_@@_caption_tl
669 \tl_new:N \l_@@_short_caption_tl
670 \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).

```

671 \bool_new:N \l_@@_caption_above_bool

```

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

```

672 \bool_new:N \l_@@_colortbl_like_bool

```

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

```

673 \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`).

```

674 \dim_new:N \l_@@_cell_space_top_limit_dim
675 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

676 \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.

```

677 \dim_new:N \l_@@_xdots_inter_dim
678 \hook_gput_code:nnn { begindocument } { . }
679 { \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).

```

680 \dim_new:N \l_@@_xdots_shorten_start_dim
681 \dim_new:N \l_@@_xdots_shorten_end_dim
682 \hook_gput_code:nnn { begindocument } { . }
683 {
684     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
685     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
686 }

```

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.

```

687 \dim_new:N \l_@@_xdots_radius_dim
688 \hook_gput_code:nnn { begindocument } { . }
689 { \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.

```

690 \tl_new:N \l_@@_xdots_line_style_tl
691 \tl_const:Nn \c_@@_standard_tl { standard }
692 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

```

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

```

693 \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).

```

694 \tl_new:N \l_@@_baseline_tl
695 \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`).

```

696 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

697 \bool_new:N \l_@@_parallelize_diags_bool
698 \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.

```

699 \clist_new:N \l_@@_corners_clist

```

```

700 \dim_new:N \l_@@_notes_above_space_dim
701 \hook_gput_code:nnn { begindocument } { . }
702 { \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.).

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

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

```
704 \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).

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

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

```
706 \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.

```
707 \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”.

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

```
709 \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).

```
710 \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`).

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

```
712 \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`.

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

```
714 \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.

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

```
716 \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 “:”.

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

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

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

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

```

719 \bool_new:N \l_@@_delimiters_max_width_bool

720 \keys_define:nn { NiceMatrix / xdots }
721 {
722   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
723   horizontal-labels .default:n = true ,
724   line-style .code:n =
725   {
726     \bool_lazy_or:nnTF
727     { \cs_if_exist_p:N \tikzpicture }
728     { \str_if_eq_p:nn { #1 } { standard } }
729     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
730     { \@@_error:n { bad-option-for-line-style } }
731   } ,
732   line-style .value_required:n = true ,
733   color .tl_set:N = \l_@@_xdots_color_tl ,
734   color .value_required:n = true ,
735   shorten .code:n =
736   { \hook_gput_code:nnn { begindocument } { . }
737     {
738       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
739       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
740     } ,
741     shorten-start .code:n =
742     { \hook_gput_code:nnn { begindocument } { . }
743       { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } } ,
744     shorten-end .code:n =
745     { \hook_gput_code:nnn { begindocument } { . }
746       { \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.

```

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

```

758   down .tl_set:N = \l_@@_xdots_down_tl ,
759   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).

```

760   draw-first .code:n = \prg_do_nothing: ,
761   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
762 }

```

```

763 \keys_define:nn { NiceMatrix / rules }
764 {
765   color .tl_set:N = \l_@@_rules_color_tl ,
766   color .value_required:n = true ,
767   width .dim_set:N = \arrayrulewidth ,
768   width .value_required:n = true ,
769   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
770 }

```

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

```

771 \keys_define:nn { NiceMatrix / Global }
772 {
773   custom-line .code:n = \@@_custom_line:n { #1 } ,
774   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
775   rules .value_required:n = true ,
776   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
777   standard-cline .default:n = true ,
778   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
779   cell-space-top-limit .value_required:n = true ,
780   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
781   cell-space-bottom-limit .value_required:n = true ,
782   cell-space-limits .meta:n =
783   {
784     cell-space-top-limit = #1 ,
785     cell-space-bottom-limit = #1 ,
786   } ,
787   cell-space-limits .value_required:n = true ,
788   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
789   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
790   light-syntax .default:n = true ,
791   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
792   end-of-row .value_required:n = true ,
793   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
794   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
795   last-row .int_set:N = \l_@@_last_row_int ,
796   last-row .default:n = -1 ,
797   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
798   code-for-first-col .value_required:n = true ,
799   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
800   code-for-last-col .value_required:n = true ,
801   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
802   code-for-first-row .value_required:n = true ,
803   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
804   code-for-last-row .value_required:n = true ,
805   hlines .clist_set:N = \l_@@_hlines_clist ,
806   vlines .clist_set:N = \l_@@_vlines_clist ,
807   hlines .default:n = all ,
808   vlines .default:n = all ,
809   vlines-in-sub-matrix .code:n =
810   {
811     \tl_if_single_token:nTF { #1 }
812     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
813     { \@@_error:n { One-letter-allowed } }
814   } ,
815   vlines-in-sub-matrix .value_required:n = true ,
816   hvlines .code:n =
817   {
818     \bool_set_true:N \l_@@_hvlines_bool
819     \clist_set:Nn \l_@@_vlines_clist { all }
820     \clist_set:Nn \l_@@_hlines_clist { all }
821   } ,
822   hvlines-except-borders .code:n =

```

```

823 {
824   \clist_set:Nn \l_@@_vlines_clist { all }
825   \clist_set:Nn \l_@@_hlines_clist { all }
826   \bool_set_true:N \l_@@_hvlines_bool
827   \bool_set_true:N \l_@@_except_borders_bool
828 } ,
829 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.

```

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

```

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

```

853 \keys_define:nn { NiceMatrix / Env }
854 {
855   corners .clist_set:N = \l_@@_corners_clist ,
856   corners .default:n = { NW , SW , NE , SE } ,
857   code-before .code:n =
858     {
859       \tl_if_empty:nF { #1 }
860       {
861         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
862         \bool_set_true:N \l_@@_code_before_bool
863       }
864     } ,
865   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}`.

```

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

```

```

875 columns-width .value_required:n = true ,
876 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.

```

877 \legacy_if:nF { measuring@ }
878 {
879   \str_set:Nx \l_tmpa_str { #1 }
880   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
881     { \@@_error:nn { Duplicate-name } { #1 } }
882     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
883   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
884 } ,
885 name .value_required:n = true ,
886 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
887 code-after .value_required:n = true ,
888 colortbl-like .code:n =
889   \bool_set_true:N \l_@@_colortbl_like_bool
890   \bool_set_true:N \l_@@_code_before_bool ,
891 colortbl-like .value_forbidden:n = true
892 }
893 \keys_define:nn { NiceMatrix / notes }
894 {
895   para .bool_set:N = \l_@@_notes_para_bool ,
896   para .default:n = true ,
897   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
898   code-before .value_required:n = true ,
899   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
900   code-after .value_required:n = true ,
901   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
902   bottomrule .default:n = true ,
903   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
904   style .value_required:n = true ,
905   label-in-tabular .code:n =
906     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
907   label-in-tabular .value_required:n = true ,
908   label-in-list .code:n =
909     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
910   label-in-list .value_required:n = true ,
911   enumitem-keys .code:n =
912     {
913       \hook_gput_code:nnn { begindocument } { . }
914       {
915         \IfPackageLoadedTF { enumitem }
916           { \setlist* [ tabularnotes ] { #1 } }
917           { }
918       }
919     } ,
920   enumitem-keys .value_required:n = true ,
921   enumitem-keys-para .code:n =
922     {
923       \hook_gput_code:nnn { begindocument } { . }
924       {
925         \IfPackageLoadedTF { enumitem }
926           { \setlist* [ tabularnotes* ] { #1 } }
927           { }
928       }
929     } ,
930   enumitem-keys-para .value_required:n = true ,
931   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
932   detect-duplicates .default:n = true ,
933   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
934 }

```

```

935 \keys_define:nn { NiceMatrix / delimiters }
936 {
937   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
938   max-width .default:n = true ,
939   color .tl_set:N = \l_@@_delimiters_color_tl ,
940   color .value_required:n = true ,
941 }

```

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

```

942 \keys_define:nn { NiceMatrix }
943 {
944   NiceMatrixOptions .inherit:n =
945     { NiceMatrix / Global } ,
946   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
947   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
948   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
949   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
950   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
951   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
952   CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
953   NiceMatrix .inherit:n =
954     {
955       NiceMatrix / Global ,
956       NiceMatrix / Env ,
957     } ,
958   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
959   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
960   NiceTabular .inherit:n =
961     {
962       NiceMatrix / Global ,
963       NiceMatrix / Env
964     } ,
965   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
966   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
967   NiceTabular / notes .inherit:n = NiceMatrix / notes ,
968   NiceArray .inherit:n =
969     {
970       NiceMatrix / Global ,
971       NiceMatrix / Env ,
972     } ,
973   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
974   NiceArray / rules .inherit:n = NiceMatrix / rules ,
975   pNiceArray .inherit:n =
976     {
977       NiceMatrix / Global ,
978       NiceMatrix / Env ,
979     } ,
980   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
981   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
982 }

```

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

```

983 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
984 {
985   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
986   delimiters / color .value_required:n = true ,
987   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
988   delimiters / max-width .default:n = true ,
989   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
990   delimiters .value_required:n = true ,

```

```

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

```

999 renew-matrix .code:n = \@@_renew_matrix: ,
1000 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}`.

```

1001 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.

```

1002 columns-width .code:n =
1003   \tl_if_eq:nnTF { #1 } { auto }
1004     { \@@_error:n { Option-auto-for-columns-width } }
1005     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1006 allow-duplicate-names .code:n =
1007   \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1008 allow-duplicate-names .value_forbidden:n = true ,
1009 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1010 notes .value_required:n = true ,
1011 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1012 sub-matrix .value_required:n = true ,
1013 matrix / columns-type .code:n =
1014   \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1015 matrix / columns-type .value_required:n = true ,
1016 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1017 caption-above .default:n = true ,
1018 unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1019 }

```

`\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.

```

1020 \NewDocumentCommand \NiceMatrixOptions { m }
1021 { \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.

```

1022 \keys_define:nn { NiceMatrix / NiceMatrix }
1023 {
1024   last-col .code:n = \tl_if_empty:nTF {#1}
1025     {
1026       \bool_set_true:N \l_@@_last_col_without_value_bool
1027       \int_set:Nn \l_@@_last_col_int { -1 }
1028     }
1029     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1030   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1031   columns-type .value_required:n = true ,

```

```

1032   l .meta:n = { columns-type = l } ,
1033   r .meta:n = { columns-type = r } ,
1034   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1035   delimiters / color .value_required:n = true ,
1036   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1037   delimiters / max-width .default:n = true ,
1038   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1039   delimiters .value_required:n = true ,
1040   small .bool_set:N = \l_@@_small_bool ,
1041   small .value_forbidden:n = true ,
1042   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1043 }

```

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

```

1044 \keys_define:nn { NiceMatrix / NiceArray }
1045 {

```

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.

```

1046   small .bool_set:N = \l_@@_small_bool ,
1047   small .value_forbidden:n = true ,
1048   last-col .code:n = \tl_if_empty:nF { #1 }
1049     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1050     \int_zero:N \l_@@_last_col_int ,
1051   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1052   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1053   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1054 }

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

```

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

```

1074 \keys_define:nn { NiceMatrix / NiceTabular }
1075 {

```

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.

```

1076   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1077     \bool_set_true:N \l_@@_width_used_bool ,
1078   width .value_required:n = true ,
1079   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
1080   rounded-corners .default:n = 4 pt ,

```



```

1081 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1082 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1083 tabularnote .value_required:n = true ,
1084 caption .tl_set:N = \l_@@_caption_tl ,
1085 caption .value_required:n = true ,
1086 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1087 short-caption .value_required:n = true ,
1088 label .tl_set:N = \l_@@_label_tl ,
1089 label .value_required:n = true ,
1090 last-col .code:n = \tl_if_empty:nF {#1}
1091 { \@@_error:n { last-col~non-empty~for~NiceArray } }
1092 \int_zero:N \l_@@_last_col_int ,
1093 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1094 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1095 unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1096 }

```

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}`).

```

1097 \cs_new_protected:Npn \@@_cell_begin:w
1098 {

```

`\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.

```

1099 \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).

```

1100 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

1101 \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.

```

1102 \int_compare:nNnT \c@jCol = 1
1103 { \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).

```

1104 \hbox_set:Nw \l_@@_cell_box
1105 \bool_if:NF \l_@@_NiceTabular_bool
1106 {
1107   \c_math_toggle_token
1108   \bool_if:NT \l_@@_small_bool \scriptstyle
1109 }
1110 \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.

```

1111 \int_compare:nNnTF \c@iRow = 0
1112 {
1113   \int_compare:nNnT \c@jCol > 0
1114   {

```

```

1115         \l_@@_code_for_first_row_tl
1116         \xglobal \colorlet { nicematrix-first-row } { . }
1117     }
1118 }
1119 {
1120     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1121     {
1122         \l_@@_code_for_last_row_tl
1123         \xglobal \colorlet { nicematrix-last-row } { . }
1124     }
1125 }
1126 }

```

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.

```

1127 \cs_new_protected:Npn \@@_begin_of_row:
1128 {
1129     \int_gincr:N \c@iRow
1130     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1131     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1132     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1133     \pgfpicture
1134     \pgfrememberpicturepositiononpagetrue
1135     \pgfcoordinate
1136     { \@@_env: - row - \int_use:N \c@iRow - base }
1137     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1138     \str_if_empty:NF \l_@@_name_str
1139     {
1140         \pgfnodelalias
1141         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1142         { \@@_env: - row - \int_use:N \c@iRow - base }
1143     }
1144     \endpgfpicture
1145 }

```

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.

```

1146 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1147 {
1148     \int_compare:nNnTF \c@iRow = 0
1149     {
1150         \dim_gset:Nn \g_@@_dp_row_zero_dim
1151         { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1152         \dim_gset:Nn \g_@@_ht_row_zero_dim
1153         { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1154     }
1155     {
1156         \int_compare:nNnT \c@iRow = 1
1157         {
1158             \dim_gset:Nn \g_@@_ht_row_one_dim
1159             { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1160         }
1161     }
1162 }
1163 \cs_new_protected:Npn \@@_rotate_cell_box:
1164 {
1165     \box_rotate:Nn \l_@@_cell_box { 90 }

```

```

1166 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1167 {
1168     \vbox_set_top:Nn \l_@@_cell_box
1169     {
1170         \vbox_to_zero:n { }
1171         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1172         \box_use:N \l_@@_cell_box
1173     }
1174 }
1175 \bool_gset_false:N \g_@@_rotate_bool
1176 }
1177 \cs_new_protected:Npn \@@_adjust_size_box:
1178 {
1179     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1180     {
1181         \box_set_wd:Nn \l_@@_cell_box
1182         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1183         \dim_gzero:N \g_@@_blocks_wd_dim
1184     }
1185     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1186     {
1187         \box_set_dp:Nn \l_@@_cell_box
1188         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1189         \dim_gzero:N \g_@@_blocks_dp_dim
1190     }
1191     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1192     {
1193         \box_set_ht:Nn \l_@@_cell_box
1194         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1195         \dim_gzero:N \g_@@_blocks_ht_dim
1196     }
1197 }
1198 \cs_new_protected:Npn \@@_cell_end:
1199 {
1200     \@@_math_toggle_token:
1201     \hbox_set_end:
1202     \@@_cell_end_i:
1203 }
1204 \cs_new_protected:Npn \@@_cell_end_i:
1205 {

```

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.

```

1206 \g_@@_cell_after_hook_tl
1207 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1208 \@@_adjust_size_box:
1209 \box_set_ht:Nn \l_@@_cell_box
1210 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1211 \box_set_dp:Nn \l_@@_cell_box
1212 { \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”).

```

1213 \dim_gset:Nn \g_@@_max_cell_width_dim
1214 { \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”.

```

1215 \@@_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.

```

1216 \bool_if:NTF \g_@@_empty_cell_bool
1217 { \box_use_drop:N \l_@@_cell_box }
1218 {
1219     \bool_lazy_or:nnTF
1220     \g_@@_not_empty_cell_bool
1221     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1222     \@@_node_for_cell:
1223     { \box_use_drop:N \l_@@_cell_box }
1224 }
1225 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1226 \bool_gset_false:N \g_@@_empty_cell_bool
1227 \bool_gset_false:N \g_@@_not_empty_cell_bool
1228 }

```

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`).

```

1229 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1230 {
1231     \@@_math_toggle_token:
1232     \hbox_set_end:
1233     \bool_if:NF \g_@@_rotate_bool
1234     {
1235         \hbox_set:Nn \l_@@_cell_box
1236         {
1237             \makebox [ \l_@@_col_width_dim ] [ s ]
1238             { \hbox_unpack_drop:N \l_@@_cell_box }
1239         }
1240     }
1241     \@@_cell_end_i:
1242 }

```

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

```

1243 \pgfset
1244 {
1245     nicematrix / cell-node /.style =
1246     {
1247         inner~sep = \c_zero_dim ,
1248         minimum~width = \c_zero_dim
1249     }
1250 }
1251 \cs_new_protected:Npn \@@_node_for_cell:
1252 {

```

```

1253 \pgfpicture
1254 \pgfsetbaseline \c_zero_dim
1255 \pgfrememberpicturepositiononpagetrue
1256 \pgfset { nicematrix / cell-node }
1257 \pgfnode
1258 { rectangle }
1259 { base }
1260 {

```

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).

```

1261 \set@color
1262 \box_use_drop:N \l_@@_cell_box
1263 }
1264 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1265 { \l_@@_pgf_node_code_tl }
1266 \str_if_empty:NF \l_@@_name_str
1267 {
1268 \pgfnodealias
1269 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1270 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1271 }
1272 \endpgfpicture
1273 }

```

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.

```

1274 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1275 {
1276 \cs_new_protected:Npn \@@_patch_node_for_cell:
1277 {
1278 \hbox_set:Nn \l_@@_cell_box
1279 {
1280 \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1281 \hbox_overlap_left:n
1282 {
1283 \pgfsys@markposition
1284 { \@@_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.

```

1285 #1
1286 }
1287 \box_use:N \l_@@_cell_box
1288 \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1289 \hbox_overlap_left:n
1290 {
1291 \pgfsys@markposition
1292 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1293 #1
1294 }
1295 }
1296 }
1297 }

```

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

```

1298 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1299 {
1300 \@@_patch_node_for_cell:n
1301 { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1302 }
1303 { \@@_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).

```
1304 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1305 {
1306   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1307   { \g_@@_#2 _ lines _ tl }
1308   {
1309     \use:c { @@ _ draw _ #2 : nnn }
1310     { \int_use:N \c@iRow }
1311     { \int_use:N \c@jCol }
1312     { \exp_not:n { #3 } }
1313   }
1314 }

1315 \cs_new_protected:Npn \@@_array:n
1316 {
1317   \bool_if:NTF \l_@@_NiceTabular_bool
1318   { \dim_set_eq:NN \col@sep \tabcolsep }
1319   { \dim_set_eq:NN \col@sep \arraycolsep }
1320   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1321   { \cs_set_nopar:Npn \@halignto { } }
1322   { \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`.

```
1323 \@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.

1324 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1325 }
1326 \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.

```
1327 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1328 \cs_new_protected:Npn \@@_create_row_node:
1329 {
1330   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1331   {
1332     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1333     \@@_create_row_node_i:
1334   }
1335 }
```

```

1336 \cs_new_protected:Npn \@@_create_row_node_i:
1337 {

```

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

```

1338   \hbox
1339   {
1340     \bool_if:NT \l_@@_code_before_bool
1341     {
1342       \vtop
1343       {
1344         \skip_vertical:N 0.5\arrayrulewidth
1345         \pgfsys@markposition
1346         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1347         \skip_vertical:N -0.5\arrayrulewidth
1348       }
1349     }
1350     \pgfpicture
1351     \pgfrememberpicturepositiononpagetrue
1352     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1353     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1354     \str_if_empty:NF \l_@@_name_str
1355     {
1356       \pgfnodealias
1357       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1358       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1359     }
1360     \endpgfpicture
1361   }
1362 }

```

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

```

1363 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1364 \cs_new_protected:Npn \@@_everycr_i:
1365 {
1366   \int_gzero:N \c@jCol
1367   \bool_gset_false:N \g_@@_after_col_zero_bool
1368   \bool_if:NF \g_@@_row_of_col_done_bool
1369   {
1370     \@@_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).

```

1371   \tl_if_empty:NF \l_@@_hlines_clist
1372   {
1373     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1374     {
1375       \exp_args:NNx
1376       \clist_if_in:NnT
1377       \l_@@_hlines_clist
1378       { \int_eval:n { \c@iRow + 1 } }
1379     }
1380   }

```

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.

```

1381   \int_compare:nNnT \c@iRow > { -1 }
1382   {
1383     \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@`.

```

1384   { \hrule height \arrayrulewidth width \c_zero_dim }

```

```

1385         }
1386     }
1387 }
1388 }
1389 }

```

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`).

```

1390 \cs_set_protected:Npn \@@_newcolumntype #1
1391 {
1392     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1393     \peek_meaning:NTF [
1394         { \newcol@ #1 }
1395         { \newcol@ #1 [ 0 ] }
1396     }

```

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

```

1397 \cs_set_protected:Npn \@@_renew_dots:
1398 {
1399     \cs_set_eq:NN \ldots \@@_Ldots
1400     \cs_set_eq:NN \cdots \@@_Cdots
1401     \cs_set_eq:NN \vdots \@@_Vdots
1402     \cs_set_eq:NN \ddots \@@_Ddots
1403     \cs_set_eq:NN \iddots \@@_Iddots
1404     \cs_set_eq:NN \dots \@@_Ldots
1405     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1406 }

```

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

```

1407 \cs_new_protected:Npn \@@_colortbl_like:
1408 {
1409     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1410     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1411     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1412 }

```

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

```

1413 \cs_new_protected:Npn \@@_pre_array_ii:
1414 {

```

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

```

1415     \int_gzero:N \g_@@_total_X_weight_int
1416     \@@_expand_clist:N \l_@@_hlines_clist
1417     \@@_expand_clist:N \l_@@_vlines_clist

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁴.

```

1418     \IfPackageLoadedTF { booktabs }
1419     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1420     { }
1421     \box_clear_new:N \l_@@_cell_box
1422     \normalbaselines

```

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

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

```

1423 \bool_if:NT \l_@@_small_bool
1424 {
1425     \cs_set_nopar:Npn \arraystretch { 0.47 }
1426     \dim_set:Nn \arraycolsep { 1.45 pt }
1427 }

1428 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1429 {
1430     \tl_put_right:Nn \@@_begin_of_row:
1431     {
1432         \pgfsys@markposition
1433         { \@@_env: - row - \int_use:N \c@iRow - base }
1434     }
1435 }

```

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`.

```

1436 \cs_set_nopar:Npn \ialign
1437 {
1438     \IfPackageLoadedTF { colortbl }
1439     {
1440         \CT@everycr
1441         {
1442             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1443             \@@_everycr:
1444         }
1445     }
1446     { \everycr { \@@_everycr: } }
1447     \tabskip = \c_zero_skip

```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`⁵ and `\extrarowheight` (of `array`). That box is inserted (via `\@arstrut`) in the beginning of each row of the array. That's why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `\@arstrutbox` and that's why we do it in the `\ialign`.

```

1448 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1449 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1450 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1451 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1452 \dim_gzero_new:N \g_@@_ht_row_one_dim
1453 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1454 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1455 \dim_gzero_new:N \g_@@_ht_last_row_dim
1456 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1457 \dim_gzero_new:N \g_@@_dp_last_row_dim
1458 \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`.

```

1459 \cs_set_eq:NN \ialign \@@_old_ialign:
1460 \halign
1461 }

```

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

We keep in memory the old versions of `\ldots`, `\cdots`, etc. only because we use them inside `\phantom` commands in order that the new commands `\Ldots`, `\Cdots`, etc. give the same spacing (except when the option `nullify-dots` is used).

```

1462 \cs_set_eq:NN \@@_old_ldots \ldots
1463 \cs_set_eq:NN \@@_old_cdots \cdots
1464 \cs_set_eq:NN \@@_old_vdots \vdots
1465 \cs_set_eq:NN \@@_old_ddots \ddots
1466 \cs_set_eq:NN \@@_old_iddots \iddots
1467 \bool_if:NTF \l_@@_standard_cline_bool
1468 { \cs_set_eq:NN \cline \@@_standard_cline }
1469 { \cs_set_eq:NN \cline \@@_cline }
1470 \cs_set_eq:NN \Ldots \@@_Ldots
1471 \cs_set_eq:NN \Cdots \@@_Cdots
1472 \cs_set_eq:NN \Vdots \@@_Vdots
1473 \cs_set_eq:NN \Ddots \@@_Ddots
1474 \cs_set_eq:NN \Iddots \@@_Iddots
1475 \cs_set_eq:NN \Hline \@@_Hline:
1476 \cs_set_eq:NN \Hspace \@@_Hspace:
1477 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1478 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1479 \cs_set_eq:NN \Block \@@_Block:
1480 \cs_set_eq:NN \rotate \@@_rotate:
1481 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1482 \cs_set_eq:NN \dotfill \@@_dotfill:
1483 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1484 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1485 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1486 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1487 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1488 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1489 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1490 \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.

```

1491 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1492 \hook_gput_code:nnn { env / tabular / begin } { . }
1493 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }

```

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

```

1494 \tl_if_exist:NT \l_@@_note_in_caption_tl
1495 {
1496   \tl_if_empty:NF \l_@@_note_in_caption_tl
1497   {
1498     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1499     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1500   }
1501 }

```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with $n > 1$ is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of n) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```

1502 \seq_gclear:N \g_@@_multicolumn_cells_seq
1503 \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).

```

1504 \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`).

```
1505 \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.

```
1506 \int_gzero_new:N \g_@@_col_total_int
1507 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1508 \@@_renew_NC@rewrite@S:
1509 \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.

```
1510 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1511 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1512 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1513 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1514 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1515 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1516 \tl_gclear:N \g_nicematrix_code_before_tl
1517 \tl_gclear:N \g_@@_pre_code_before_tl
1518 }
```

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

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

```
1519 \cs_new_protected:Npn \@@_pre_array:
1520 {
1521 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1522 \int_gzero_new:N \c@iRow
1523 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1524 \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).

```
1525 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1526 {
1527 \bool_set_true:N \l_@@_last_row_without_value_bool
1528 \bool_if:NT \g_@@_aux_found_bool
1529 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1530 }
1531 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1532 {
1533 \bool_if:NT \g_@@_aux_found_bool
1534 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1535 }
```

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”.

```
1536 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1537 {
1538 \tl_put_right:Nn \@@_update_for_first_and_last_row:
```

```

1539     {
1540         \dim_gset:Nn \g_@@_ht_last_row_dim
1541         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1542         \dim_gset:Nn \g_@@_dp_last_row_dim
1543         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1544     }
1545 }

1546 \seq_gclear:N \g_@@_cols_vlism_seq
1547 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1548 \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.

```

1549 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1550 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1551 \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.

```

1552 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value -2 is important.

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

```

1553 \@@_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.

```

1554 \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]`).

```

1555 \dim_zero_new:N \l_@@_left_delim_dim
1556 \dim_zero_new:N \l_@@_right_delim_dim
1557 \bool_if:NTF \g_@@_NiceArray_bool
1558 {
1559     \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1560     \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1561 }
1562 {

```

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

```

1563 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1564 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1565 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1566 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1567 }

```

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).

```

1568 \hbox_set:Nw \l_@@_the_array_box

```

```

1569 \skip_horizontal:N \l_@@_left_margin_dim
1570 \skip_horizontal:N \l_@@_extra_left_margin_dim
1571 \c_math_toggle_token
1572 \bool_if:NTF \l_@@_light_syntax_bool
1573 { \use:c { @@-light-syntax } }
1574 { \use:c { @@-normal-syntax } }
1575 }

```

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

```

1576 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1577 {
1578   \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1579   \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...

```

1580 \@@_pre_array:
1581 }

```

9 The `\CodeBefore`

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

```

1582 \cs_new_protected:Npn \@@_pre_code_before:
1583 {

```

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).

```

1584 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1585 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1586 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1587 \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.

```

1588 \pgfsys@markposition { \@@_env: - position }
1589 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1590 \pgfpicture
1591 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1592 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1593 {
1594   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1595   \pgfcoordinate { \@@_env: - row - ##1 }
1596   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1597 }

```

Now, the recreation of the `col` nodes.

```

1598 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1599 {
1600   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1601   \pgfcoordinate { \@@_env: - col - ##1 }
1602   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1603 }

```

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

```
1604 \@@_create_diag_nodes:
```

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

```
1605 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1606 \endpgfpicture
```

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

```
1607 \@@_create_blocks_nodes:
1608 \IfPackageLoadedTF { tikz }
1609 {
1610     \tikzset
1611     {
1612         every~picture / .style =
1613         { overlay , name~prefix = \@@_env: - }
1614     }
1615 }
1616 { }
1617 \cs_set_eq:NN \cellcolor \@@_cellcolor
1618 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1619 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1620 \cs_set_eq:NN \rowcolor \@@_rowcolor
1621 \cs_set_eq:NN \rowcolors \@@_rowcolors
1622 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1623 \cs_set_eq:NN \arraycolor \@@_arraycolor
1624 \cs_set_eq:NN \columncolor \@@_columncolor
1625 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1626 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1627 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1628 }

1629 \cs_new_protected:Npn \@@_exec_code_before:
1630 {
1631     \seq_gclear_new:N \g_@@_colors_seq
1632     \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1633     \group_begin:
```

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

```
1634 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
```

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

```
1635 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1636 {
1637     \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1638     \@@_rescan_for_spanish:N \l_@@_code_before_tl
1639 }
```

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`.

```
1640 \exp_last_unbraced:N \@@_CodeBefore_keys:
1641 \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.

```

1642     \@@_actually_color:
1643     \l_@@_code_before_tl
1644     \q_stop
1645     \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1646     \group_end:
1647     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1648     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1649   }

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

1660 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1661 {
1662   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1663   \@@_CodeBefore:w
1664 }

```

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

```

1665 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1666 {
1667   \bool_if:NT \g_@@_aux_found_bool
1668   {
1669     \@@_pre_code_before:
1670     #1
1671   }
1672 }

```

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.

```

1673 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1674 {
1675   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1676   {
1677     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1678     \pgfcoordinate { \@@_env: - row - ##1 - base }
1679     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1680     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1681     {
1682       \cs_if_exist:cT
1683       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1684       {
1685         \pgfsys@getposition
1686         { \@@_env: - ##1 - #####1 - NW }
1687         \@@_node_position:
1688         \pgfsys@getposition

```

```

1689         { \@@_env: - ##1 - #####1 - SE }
1690         \@@_node_position_i:
1691         \@@_pgf_rect_node:nnn
1692         { \@@_env: - ##1 - #####1 }
1693         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1694         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1695     }
1696 }
1697 }
1698 \int_step_inline:nn \c@iRow
1699 {
1700     \pgfnodealias
1701     { \@@_env: - ##1 - last }
1702     { \@@_env: - ##1 - \int_use:N \c@jCol }
1703 }
1704 \int_step_inline:nn \c@jCol
1705 {
1706     \pgfnodealias
1707     { \@@_env: - last - ##1 }
1708     { \@@_env: - \int_use:N \c@iRow - ##1 }
1709 }
1710 \@@_create_extra_nodes:
1711 }

```

```

1712 \cs_new_protected:Npn \@@_create_blocks_nodes:
1713 {
1714     \pgfpicture
1715     \pgf@relevantforpicturesizefalse
1716     \pgfrememberpicturepositiononpagetrue
1717     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1718     { \@@_create_one_block_node:nnnnn ##1 }
1719     \endpgfpicture
1720 }

```

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

```

1721 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1722 {
1723     \tl_if_empty:nF { #5 }
1724     {
1725         \@@_qpoint:n { col - #2 }
1726         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1727         \@@_qpoint:n { #1 }
1728         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1729         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1730         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1731         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1732         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1733         \@@_pgf_rect_node:nnnnn
1734         { \@@_env: - #5 }
1735         { \dim_use:N \l_tmpa_dim }
1736         { \dim_use:N \l_tmpb_dim }
1737         { \dim_use:N \l_@@_tmpc_dim }
1738         { \dim_use:N \l_@@_tmpd_dim }
1739     }
1740 }

```

```

1741 \cs_new_protected:Npn \@@_patch_for_revtext:

```

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).


```

1742 {
1743   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1744   \cs_set_eq:NN \insert@column \insert@column@array
1745   \cs_set_eq:NN \@classx \@classx@array
1746   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1747   \cs_set_eq:NN \@arraycr \@arraycr@array
1748   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1749   \cs_set_eq:NN \array \array@array
1750   \cs_set_eq:NN \@array \@array@array
1751   \cs_set_eq:NN \@tabular \@tabular@array
1752   \cs_set_eq:NN \@mkpream \@mkpream@array
1753   \cs_set_eq:NN \endarray \endarray@array
1754   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1755   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1756 }

```

10 The environment `{NiceArrayWithDelims}`

```

1757 \NewDocumentEnvironment { NiceArrayWithDelims }
1758 { m m O { } m ! O { } t \CodeBefore }
1759 {
1760   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1761   \@@_provide_pgfsyspdfmark:
1762   \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.

```

1763   \bgroup

1764   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1765   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1766   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1767   \int_gzero:N \g_@@_block_box_int
1768   \dim_zero:N \g_@@_width_last_col_dim
1769   \dim_zero:N \g_@@_width_first_col_dim
1770   \bool_gset_false:N \g_@@_row_of_col_done_bool
1771   \str_if_empty:NT \g_@@_name_env_str
1772     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1773   \bool_if:NTF \l_@@_NiceTabular_bool
1774     \mode_leave_vertical:
1775     \@@_test_if_math_mode:
1776   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1777   \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.

```

1778   \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.

```

1779   \cs_if_exist:NT \tikz@library@external@loaded

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1780 {
1781   \tikzexternaldisable
1782   \cs_if_exist:NT \ifstandalone
1783     { \tikzset { external / optimize = false } }
1784 }

```

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

```

1785 \int_gincr:N \g_@@_env_int
1786 \bool_if:NF \l_@@_block_auto_columns_width_bool
1787 { \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`).

```

1788 \seq_gclear:N \g_@@_blocks_seq
1789 \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`.

```

1790 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1791 \seq_gclear:N \g_@@_pos_of_xdots_seq
1792 \tl_gclear_new:N \g_@@_code_before_tl
1793 \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.

```

1794 \bool_gset_false:N \g_@@_aux_found_bool
1795 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1796 {
1797   \bool_gset_true:N \g_@@_aux_found_bool
1798   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1799 }

```

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.

```

1800 \tl_gclear:N \g_@@_aux_tl
1801 \tl_if_empty:NF \g_@@_code_before_tl
1802 {
1803   \bool_set_true:N \l_@@_code_before_bool
1804   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1805 }
1806 \tl_if_empty:NF \g_@@_pre_code_before_tl
1807 { \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`.

```

1808 \bool_if:NTF \g_@@_NiceArray_bool
1809 { \keys_set:nn { NiceMatrix / NiceArray } }
1810 { \keys_set:nn { NiceMatrix / pNiceArray } }
1811 { #3 , #5 }

1812 \@@_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:.`

```

1813 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1814 }

```

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

```

1815 {

```

```

1816 \bool_if:NTF \l_@@_light_syntax_bool
1817 { \use:c { end @@-light-syntax } }
1818 { \use:c { end @@-normal-syntax } }
1819 \c_math_toggle_token
1820 \skip_horizontal:N \l_@@_right_margin_dim
1821 \skip_horizontal:N \l_@@_extra_right_margin_dim
1822 \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.

```

1823 \bool_if:NT \l_@@_width_used_bool
1824 {
1825   \int_compare:nNnT \g_@@_total_X_weight_int = 0
1826   { \@@_error_or_warning:n { width~without~X~columns } }
1827 }

```

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 .

```

1828 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1829 {
1830   \tl_gput_right:Nx \g_@@_aux_tl
1831   {
1832     \bool_set_true:N \l_@@_X_columns_aux_bool
1833     \dim_set:Nn \l_@@_X_columns_dim
1834     {
1835       \dim_compare:nNnTF
1836       {
1837         \dim_abs:n
1838         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1839       }
1840       <
1841       { 0.001 pt }
1842       { \dim_use:N \l_@@_X_columns_dim }
1843       {
1844         \dim_eval:n
1845         {
1846           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1847           / \int_use:N \g_@@_total_X_weight_int
1848           + \l_@@_X_columns_dim
1849         }
1850       }
1851     }
1852   }
1853 }

```

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

```

1854 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1855 {
1856   \bool_if:NF \l_@@_last_row_without_value_bool
1857   {
1858     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1859     {
1860       \@@_error:n { Wrong~last~row }
1861       \int_gset_eq:NN \l_@@_last_row_int \c_iRow
1862     }
1863   }
1864 }

```

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

```

1865 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1866 \bool_if:nTF \g_@@_last_col_found_bool
1867 { \int_gdecr:N \c@jCol }
1868 {
1869   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1870   { \@@_error:n { last~col~not~used } }
1871 }

```

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

```

1872 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1873 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 84).

```

1874 \int_compare:nNnT \l_@@_first_col_int = 0
1875 {
1876   \skip_horizontal:N \col@sep
1877   \skip_horizontal:N \g_@@_width_first_col_dim
1878 }

```

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

```

1879 \bool_if:nTF \g_@@_NiceArray_bool
1880 {
1881   \str_case:VnF \l_@@_baseline_tl
1882   {
1883     b \@@_use_arraybox_with_notes_b:
1884     c \@@_use_arraybox_with_notes_c:
1885   }
1886   \@@_use_arraybox_with_notes:
1887 }

```

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

```

1888 {
1889   \int_compare:nNnTF \l_@@_first_row_int = 0
1890   {
1891     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1892     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1893   }
1894   { \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

```

1895   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1896   {
1897     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1898     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1899   }
1900   { \dim_zero:N \l_tmpb_dim }
1901   \hbox_set:Nn \l_tmpa_box
1902   {
1903     \c_math_toggle_token
1904     \@@_color:V \l_@@_delimiters_color_tl
1905     \exp_after:wN \left \g_@@_left_delim_tl

```

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

⁹A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```

1906         \vcenter
1907         {

```

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.

```

1908             \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1909             \hbox
1910             {
1911                 \bool_if:NTF \l_@@_NiceTabular_bool
1912                 { \skip_horizontal:N -\tabcolsep }
1913                 { \skip_horizontal:N -\arraycolsep }
1914                 \@@_use_arraybox_with_notes_c:
1915                 \bool_if:NTF \l_@@_NiceTabular_bool
1916                 { \skip_horizontal:N -\tabcolsep }
1917                 { \skip_horizontal:N -\arraycolsep }
1918             }

```

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

```

1919             \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1920         }

```

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.

```

1921         \@@_color:V \l_@@_delimiters_color_tl
1922         \exp_after:wN \right \g_@@_right_delim_tl
1923         \c_math_toggle_token
1924     }

```

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.

```

1925         \bool_if:NTF \l_@@_delimiters_max_width_bool
1926         {
1927             \@@_put_box_in_flow_bis:nn
1928             \g_@@_left_delim_tl \g_@@_right_delim_tl
1929         }
1930         \@@_put_box_in_flow:
1931     }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 85).

```

1932     \bool_if:NT \g_@@_last_col_found_bool
1933     {
1934         \skip_horizontal:N \g_@@_width_last_col_dim
1935         \skip_horizontal:N \col@sep
1936     }
1937     \bool_if:NF \l_@@_Matrix_bool
1938     {
1939         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1940         { \@@_warning_gredirect_none:n { columns-not-used } }
1941     }
1942     \@@_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.

```

1943     \egroup

```

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

```

1944     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1945     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1946     \iow_now:Nx \@mainaux
1947     {
1948         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1949         { \exp_not:V \g_@@_aux_tl }
1950     }
1951     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

```

```

1952   \bool_if:NT \c_@@_footnote_bool \endsavenotes
1953 }

```

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.

```

1954 \cs_new_protected:Npn \@@_transform_preamble:
1955 {

```

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

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

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

```

1956   \group_begin:

```

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

```

1957   \bool_if:NF \l_@@_Matrix_bool
1958   {
1959     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1960     \@@_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).

```

1961     \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).

```

1962     \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1963     \@tempswatru

```

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

```

1964     \@whilesw \if@tempwa \fi { \@tempwafalse \the \NC@list }

```

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

```

1965     \int_gzero:N \c@jCol
1966     \tl_gclear:N \g_@@_preamble_tl

```

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

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

```

1967     \bool_gset_false:N \g_tmpb_bool
1968     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1969     {
1970         \tl_gset:Nn \g_@@_preamble_tl
1971         { ! { \skip_horizontal:N \arrayrulewidth } }
1972     }
1973     {
1974         \clist_if_in:NnT \l_@@_vlines_clist 1
1975         {
1976             \tl_gset:Nn \g_@@_preamble_tl
1977             { ! { \skip_horizontal:N \arrayrulewidth } }
1978         }
1979     }

```

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

```

1980     \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1981     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

1982     \int_zero:N \l_tmpa_int

```

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

```

1983     \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1984     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1985 }

```

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

```

1986     \bool_if:NT \l_@@_colortbl_like_bool
1987     {
1988         \regex_replace_all:NnN
1989         \c_@@_columncolor_regex
1990         { \c { @@_columncolor_preamble } }
1991         \g_@@_preamble_tl
1992     }

```

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

```

1993     \group_end:

```

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

```

1994     \bool_lazy_or:nnT
1995     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1996     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1997     { \bool_gset_false:N \g_@@_NiceArray_bool }

```

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

```

1998     \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).

```

1999     \int_compare:nNnTF \l_@@_first_col_int = 0
2000     { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
2001     {
2002         \bool_lazy_all:nT
2003         {
2004             \g_@@_NiceArray_bool
2005             { \bool_not_p:n \l_@@_NiceTabular_bool }
2006             { \tl_if_empty_p:N \l_@@_vlines_clist }

```

```

2007         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2008     }
2009     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
2010 }
2011 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2012 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
2013 {
2014     \bool_lazy_all:nT
2015     {
2016         \g_@@_NiceArray_bool
2017         { \bool_not_p:n \l_@@_NiceTabular_bool }
2018         { \tl_if_empty_p:N \l_@@_vlines_clist }
2019         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2020     }
2021     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
2022 }

```

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`).

```

2023     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2024     {
2025         \tl_gput_right:Nn \g_@@_preamble_tl
2026         { > { \@@_error_too_much_cols: } 1 }
2027     }
2028 }

```

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

```

2029 \cs_new_protected:Npn \@@_patch_preamble:n #1
2030 {
2031     \str_case:nnF { #1 }
2032     {
2033         c      { \@@_patch_preamble_i:n #1 }
2034         l      { \@@_patch_preamble_i:n #1 }
2035         r      { \@@_patch_preamble_i:n #1 }
2036         >      { \@@_patch_preamble_xiv:n }
2037         !      { \@@_patch_preamble_ii:nn #1 }
2038         @      { \@@_patch_preamble_ii:nn #1 }
2039         |      { \@@_patch_preamble_iii:n #1 }
2040         p      { \@@_patch_preamble_iv:n #1 }
2041         b      { \@@_patch_preamble_iv:n #1 }
2042         m      { \@@_patch_preamble_iv:n #1 }
2043         \@@_V: { \@@_patch_preamble_v:n }
2044         V      { \@@_patch_preamble_v:n }
2045         \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
2046         \@@_W: { \@@_patch_preamble_vi:nnnn { \@@_special_W: } #1 }
2047         \@@_S: { \@@_patch_preamble_vii:n }
2048         (      { \@@_patch_preamble_viii:nn #1 }
2049         [      { \@@_patch_preamble_viii:nn #1 }
2050         \{      { \@@_patch_preamble_viii:nn #1 }
2051         \left  { \@@_patch_preamble_viii:nn }
2052         )      { \@@_patch_preamble_ix:nn #1 }
2053         ]      { \@@_patch_preamble_ix:nn #1 }
2054         \}      { \@@_patch_preamble_ix:nn #1 }
2055         \right { \@@_patch_preamble_ix:nn }
2056         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}`.

```

2057     \@@_X      { \@@_patch_preamble_x:n }
2058     \q_stop { }
2059 }

```



```

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

```

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).

```

2069 {
2070   \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2071   {
2072     \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2073     \@@_patch_preamble:n
2074   }
2075   {
2076     \tl_if_eq:nnT { #1 } { S }
2077     { \@@_fatal:n { unknown~column~type~S } }
2078     { \@@_fatal:nn { unknown~column~type } { #1 } }
2079   }
2080 }
2081 }
2082 }

```

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`

```

2083 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2084 {
2085   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2086   \tl_gclear:N \g_@@_pre_cell_tl
2087   \tl_gput_right:Nn \g_@@_preamble_tl
2088   {
2089     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2090     #1
2091     < \@@_cell_end:
2092   }

```

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

```

2093   \int_gincr:N \c@jCol
2094   \@@_patch_preamble_xi:n
2095 }

```

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

```

2096 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2097 {
2098   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2099   \@@_patch_preamble:n
2100 }

```

For `|`

```

2101 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2102 {

```

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

```

2103   \int_incr:N \l_tmpa_int
2104   \@@_patch_preamble_iii_i:n
2105 }
2106 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2107 {
2108   \str_if_eq:nnTF { #1 } |
2109   { \@@_patch_preamble_iii_i:n | }
2110   {
2111     \dim_set:Nn \l_tmpa_dim
2112     {
2113       \arrayrulewidth * \l_tmpa_int
2114       + \doublerulesep * ( \l_tmpa_int - 1 )
2115     }
2116     \tl_gput_right:Nx \g_@@_preamble_tl
2117     {

```

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

```

2118       \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_tmpa_dim } } }
2119     }
2120     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2121     {
2122       \@@_vline:n
2123       {
2124         position = \int_eval:n { \c@jCol + 1 } ,
2125         multiplicity = \int_use:N \l_tmpa_int ,
2126         total-width = \dim_use:N \l_tmpa_dim
2127       }

```

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.

```

2128     }
2129     \int_zero:N \l_tmpa_int
2130     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2131     \@@_patch_preamble:n #1
2132   }
2133 }
2134 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2135 {
2136   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2137   \@@_patch_preamble:n
2138 }
2139 \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.

```

2140 \keys_define:nn { WithArrows / p-column }
2141 {
2142   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2143   r .value_forbidden:n = true ,
2144   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2145   c .value_forbidden:n = true ,
2146   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2147   l .value_forbidden:n = true ,
2148   R .code:n =
2149   \IfPackageLoadedTF { ragged2e }
2150   { \str_set:Nn \l_@@_hpos_col_str { R } }
2151   {
2152     \@@_error_or_warning:n { ragged2e-not-loaded }
2153     \str_set:Nn \l_@@_hpos_col_str { r }
2154   } ,

```

```

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

```

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

```

2182 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2183 {
2184   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2185   \@@_patch_preamble_iv_i:n
2186 }

2187 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2188 {
2189   \str_if_eq:nnTF { #1 } { [ ]
2190     { \@@_patch_preamble_iv_ii:w [ ]
2191       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2192     }
2193   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2194     { \@@_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.

```

2195 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2196 {

```

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).

```

2197   \str_set:Nn \l_@@_hpos_col_str { j }
2198   \tl_set:Nn \l_tmpa_tl { #1 }
2199   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2200   \@@_keys_p_column:V \l_tmpa_tl
2201   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2202 }

2203 \cs_new_protected:Npn \@@_keys_p_column:n #1
2204 { \keys_set_known:nn { WithArrows / p-column } { #1 } \l_tmpa_tl }
2205 \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`.

```

2206 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2207 {
2208   \use:x
2209   {
2210     \@@_patch_preamble_iv_v:nnnnnnnn
2211     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2212     { \dim_eval:n { #1 } }
2213     {

```

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.

```

2214       \str_if_eq:VnTF \l_@@_hpos_col_str j
2215       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2216       {
2217         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2218         { \str_lowercase:V \l_@@_hpos_col_str }
2219       }
2220     \str_case:Vn \l_@@_hpos_col_str
2221     {
2222       c { \exp_not:N \centering }
2223       l { \exp_not:N \raggedright }
2224       r { \exp_not:N \raggedleft }
2225       C { \exp_not:N \Centering }
2226       L { \exp_not:N \RaggedRight }
2227       R { \exp_not:N \RaggedLeft }
2228     }
2229   }
2230   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2231   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2232   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2233   { #2 }
2234   {
2235     \str_case:VnF \l_@@_hpos_col_str
2236     {
2237       { j } { c }
2238       { si } { c }
2239     }

```

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

```

2240       { \str_lowercase:V \l_@@_hpos_col_str }
2241     }
2242   }

```

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

```

2243   \int_gincr:N \c@jCol
2244   \@@_patch_preamble_xi:n
2245 }

```

#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*.

```

2246 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2247 {
2248   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2249   { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2250   { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2251   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2252   \tl_gclear:N \g_@@_pre_cell_tl
2253   \tl_gput_right:Nn \g_@@_preamble_tl
2254   {
2255     > {

```

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.

```

2256       \dim_set:Nn \l_@@_col_width_dim { #2 }
2257       \@@_cell_begin:w
2258       \begin { #7 } [ #1 ] { #2 }

```

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

```

2259       \everypar
2260       {
2261         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2262         \everypar { }
2263       }

```

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

```

2264       #3

```

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

```

2265       \g_@@_row_style_tl
2266       \arraybackslash
2267       #5
2268     }
2269     #8
2270     < {
2271       #6

```

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

```

2272       \@finalstrut \@arstrutbox
2273       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
2274       \end { #7 }

```

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

```

2275       #4
2276       \@@_cell_end:
2277     }
2278   }
2279 }

```

```

2280 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2281 {
2282   \peek_meaning:NT \unskip
2283   {
2284     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2285     {
2286       \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.

```

2287       \skip_horizontal:N \l_@@_col_width_dim
2288     }
2289   }
2290   #1
2291 }

```

```

2292 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2293 {
2294   \peek_meaning:NT \_siunitx_table_skip:n
2295   {
2296     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2297     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2298   }
2299   #1
2300 }

```

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.

```

2301 \cs_new_protected:Npn \@@_center_cell_box:
2302 {

```

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

```

2303   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2304   {
2305     \int_compare:nNnT
2306     { \box_ht:N \l_@@_cell_box }
2307     >

```

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).

```

2308     { \box_ht:N \strutbox }
2309     {
2310       \hbox_set:Nn \l_@@_cell_box
2311       {
2312         \box_move_down:nn
2313         {
2314           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2315             + \baselineskip ) / 2
2316         }
2317         { \box_use:N \l_@@_cell_box }
2318       }
2319     }
2320   }
2321 }

```

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

```

2322 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2323 {
2324   \str_if_eq:nnTF { #1 } { [ ] }
2325   { \@@_patch_preamble_v_i:w [ ] }
2326   { \@@_patch_preamble_v_i:w [ ] { #1 } }
2327 }
2328 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2329 { \@@_patch_preamble_v_ii:nn { #1 } }
2330 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2331 {
2332   \str_set:Nn \l_@@_vpos_col_str { p }
2333   \str_set:Nn \l_@@_hpos_col_str { j }
2334   \tl_set:Nn \l_tmpa_tl { #1 }
2335   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2336   \@@_keys_p_column:V \l_tmpa_tl
2337   \IfPackageLoadedTF { varwidth }
2338   { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2339   {

```

```

2340     \@@_error_or_warning:n { varwidth~not~loaded }
2341     \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2342   }
2343 }

```

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.

```

2344 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2345 {
2346   \str_if_eq:nnTF { #3 } { s }
2347     { \@@_patch_preamble_vi_i:nnnn { #1 } { #4 } }
2348     { \@@_patch_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2349 }

```

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.

```

2350 \cs_new_protected:Npn \@@_patch_preamble_vi_i:nnnn #1 #2
2351 {
2352   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2353   \tl_gclear:N \g_@@_pre_cell_tl
2354   \tl_gput_right:Nn \g_@@_preamble_tl
2355     {
2356       > {
2357         \dim_set:Nn \l_@@_col_width_dim { #2 }
2358         \@@_cell_begin:w
2359         \str_set:Nn \l_@@_hpos_cell_str { c }
2360       }
2361       c
2362       < {
2363         \@@_cell_end_for_w_s:
2364         #1
2365         \@@_adjust_size_box:
2366         \box_use_drop:N \l_@@_cell_box
2367       }
2368     }
2369   \int_gincr:N \c@jCol
2370   \@@_patch_preamble_xi:n
2371 }

```

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

```

2372 \cs_new_protected:Npn \@@_patch_preamble_vi_ii:nnnn #1 #2 #3 #4
2373 {
2374   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2375   \tl_gclear:N \g_@@_pre_cell_tl
2376   \tl_gput_right:Nn \g_@@_preamble_tl
2377     {
2378       > {

```

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.

```

2379         \dim_set:Nn \l_@@_col_width_dim { #4 }
2380         \hbox_set:Nw \l_@@_cell_box
2381         \@@_cell_begin:w
2382         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2383       }
2384       c
2385       < {
2386         \@@_cell_end:

```

```

2387         \hbox_set_end:
2388         % The following line is probably pointless
2389         % \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2390         #1
2391         \@@_adjust_size_box:
2392         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2393     }
2394 }

```

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

```

2395     \int_gincr:N \c@jCol
2396     \@@_patch_preamble_xi:n
2397 }

2398 \cs_new_protected:Npn \@@_special_W:
2399 {
2400     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2401     { \@@_warning:n { W-warning } }
2402 }

```

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).

```

2403 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2404 {
2405     \str_if_eq:nnTF { #1 } { [ ]
2406     { \@@_patch_preamble_vii_i:w [ ]
2407     { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2408     }

2409     \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2410     { \@@_patch_preamble_vii_ii:n { #1 } }

2411     \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2412     {
2413         \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2414         {
2415             \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2416             \tl_gclear:N \g_@@_pre_cell_tl
2417             \tl_gput_right:Nn \g_@@_preamble_tl
2418             {
2419                 > {
2420                     \@@_cell_begin:w
2421                     \keys_set:nn { siunitx } { #1 }
2422                     \siunitx_cell_begin:w
2423                 }
2424                 c
2425                 < { \siunitx_cell_end: \@@_cell_end: }
2426             }

```

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

```

2427     \int_gincr:N \c@jCol
2428     \@@_patch_preamble_xi:n
2429 }
2430 { \@@_fatal:n { Version-of-siunitx-too-old } }
2431 }

```

For (, [and \{.

```

2432 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2433 {
2434     \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.

```

2435     \int_compare:nNnTF \c@jCol = \c_zero_int
2436     {
2437         \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2438         {

```


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

```

2439         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2440         \tl_gset:Nn \g_@@_right_delim_tl { . }
2441         \@@_patch_preamble:n #2
2442     }
2443     {
2444         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2445         \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2446     }
2447 }
2448 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2449 }

2450 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2451 {
2452     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2453     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2454     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2455     {
2456         \@@_error:nn { delimiter~after~opening } { #2 }
2457         \@@_patch_preamble:n
2458     }
2459     { \@@_patch_preamble:n #2 }
2460 }

```

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}`).

```

2461 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2462 {
2463     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2464     \tl_if_in:nnTF { ) ] \} } { #2 }
2465     { \@@_patch_preamble_ix_i:nnn #1 #2 }
2466     {
2467         \tl_if_eq:nnTF { \q_stop } { #2 }
2468         {
2469             \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2470             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2471             {
2472                 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2473                 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2474                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2475                 \@@_patch_preamble:n #2
2476             }
2477         }
2478         {
2479             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2480             { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2481             \tl_gput_right:Nx \g_@@_pre_code_after_tl
2482             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2483             \@@_patch_preamble:n #2
2484         }
2485     }
2486 }

2487 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2488 {
2489     \tl_if_eq:nnTF { \q_stop } { #3 }
2490     {
2491         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2492         {

```

```

2493 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2494 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2495 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2496 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2497 }
2498 {
2499 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2500 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2501 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2502 \@@_error:nn { double~closing~delimiter } { #2 }
2503 }
2504 }
2505 {
2506 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2507 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2508 \@@_error:nn { double~closing~delimiter } { #2 }
2509 \@@_patch_preamble:n #3
2510 }
2511 }

```

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.

```

2512 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2513 {
2514 \str_if_eq:nnTF { #1 } { [ ]
2515 { \@@_patch_preamble_x_i:w [ ]
2516 { \@@_patch_preamble_x_i:w [ ] #1 }
2517 }
2518 \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2519 { \@@_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).

```

2520 \keys_define:nn { WithArrows / X-column }
2521 { 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.

```

2522 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2523 {

```

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).

```

2524 \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).

```

2525 \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.

```

2526 \int_zero_new:N \l_@@_weight_int
2527 \int_set:Nn \l_@@_weight_int { 1 }
2528 \tl_set:Nn \l_tmpa_tl { #1 }
2529 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2530 \@@_keys_p_column:V \l_tmpa_tl
2531 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2532 \int_compare:nNnT \l_@@_weight_int < 0
2533 {

```

```

2534     \@@_error_or_warning:n { negative-weight }
2535     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2536   }
2537   \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).

```

2538   \bool_if:NTF \l_@@_X_columns_aux_bool
2539   {
2540     \exp_args:Nnx
2541     \@@_patch_preamble_iv_iv:nn
2542     { \l_@@_weight_int \l_@@_X_columns_dim }
2543     { minipage }
2544   }
2545   {
2546     \tl_gput_right:Nn \g_@@_preamble_tl
2547     {
2548       > {
2549         \@@_cell_begin:w
2550         \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`.

```

2551     \NotEmpty

```

The following code will nullify the box of the cell.

```

2552     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2553     { \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`.

```

2554     \begin { minipage } { 5 cm } \arraybackslash
2555   }
2556   c
2557   < {
2558     \end { minipage }
2559     \@@_cell_end:
2560   }
2561   }
2562   \int_gincr:N \c@jCol
2563   \@@_patch_preamble_xi:n
2564 }
2565 }

```

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 `@{...}`.

```

2566 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2567 {
2568   \str_if_eq:nnTF { #1 } { < }
2569   \@@_patch_preamble_xiii:n
2570   {
2571     \str_if_eq:nnTF { #1 } { @ }
2572     \@@_patch_preamble_xv:n
2573     {
2574       \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2575       {
2576         \tl_gput_right:Nn \g_@@_preamble_tl
2577         { ! { \skip_horizontal:N \arrayrulewidth } }
2578       }
2579       {
2580         \exp_args:NNx

```

```

2581         \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2582         {
2583             \tl_gput_right:Nn \g_@@_preamble_tl
2584             { ! { \skip_horizontal:N \arrayrulewidth } }
2585         }
2586     }
2587     \@@_patch_preamble:n { #1 }
2588 }
2589 }
2590 }
2591 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2592 {
2593     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2594     \@@_patch_preamble_xi:n
2595 }

```

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.

```

2596 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2597 {
2598     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2599     {
2600         \tl_gput_right:Nn \g_@@_preamble_tl
2601         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2602     }
2603     {
2604         \exp_args:NNx
2605         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2606         {
2607             \tl_gput_right:Nn \g_@@_preamble_tl
2608             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2609         }
2610         { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2611     }
2612     \@@_patch_preamble:n
2613 }
2614 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2615 {
2616     \group_begin:
2617     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2618     \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2619     \temptokena { #2 }
2620     \@tempswatrue
2621     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2622     \tl_gclear:N \g_@@_preamble_tl
2623     \exp_after:wN \@@_patch_m_preamble:n \the \temptokena \q_stop
2624     \group_end:
2625     \tl_set_eq:NN #1 \g_@@_preamble_tl
2626 }

```

12 The redefinition of `\multicolumn`

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

```

2627 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2628 {

```

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`.

```

2629 \multispan { #1 }
2630 \begingroup
2631 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
2632 \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2633 \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

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

```

2634 \@temptokena = { #2 }
2635 \@tempswatrue
2636 \@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`.

```

2637 \tl_gclear:N \g_@@_preamble_tl
2638 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

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

```

2639 \exp_args:NV \@mkpream \g_@@_preamble_tl
2640 \@addtopreamble \@empty
2641 \endgroup

```

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

```

2642 \int_compare:nNnT { #1 } > 1
2643 {
2644   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2645   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2646   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2647   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2648   {
2649     {
2650       \int_compare:nNnTF \c@jCol = 0
2651       { \int_eval:n { \c@iRow + 1 } }
2652       { \int_use:N \c@iRow }
2653     }
2654     { \int_eval:n { \c@jCol + 1 } }
2655     {
2656       \int_compare:nNnTF \c@jCol = 0
2657       { \int_eval:n { \c@iRow + 1 } }
2658       { \int_use:N \c@iRow }
2659     }
2660     { \int_eval:n { \c@jCol + #1 } }
2661     { } % for the name of the block
2662   }
2663 }

```

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

```

2664 \cs_set:Npn \@sharp { #3 }
2665 \@arstrut
2666 \@preamble
2667 \null

```

We add some lines.

```

2668 \int_gadd:Nn \c@jCol { #1 - 1 }
2669 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2670 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2671 \ignorespaces
2672 }

```

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`.

```

2673 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2674 {
2675   \str_case:nnF { #1 }
2676   {
2677     c { \@@_patch_m_preamble_i:n #1 }
2678     l { \@@_patch_m_preamble_i:n #1 }
2679     r { \@@_patch_m_preamble_i:n #1 }
2680     > { \@@_patch_m_preamble_ii:nn #1 }
2681     ! { \@@_patch_m_preamble_ii:nn #1 }
2682     @ { \@@_patch_m_preamble_ii:nn #1 }
2683     | { \@@_patch_m_preamble_iii:n #1 }
2684     p { \@@_patch_m_preamble_iv:nnn t #1 }
2685     m { \@@_patch_m_preamble_iv:nnn c #1 }
2686     b { \@@_patch_m_preamble_iv:nnn b #1 }
2687     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2688     \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2689     \q_stop { }
2690   }
2691   {
2692     \tl_if_eq:nnT { #1 } { S }
2693     { \@@_fatal:n { unknown-column~type~S } }
2694     { \@@_fatal:nn { unknown-column~type } { #1 } }
2695   }
2696 }

```

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

```

2697 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2698 {
2699   \tl_gput_right:Nn \g_@@_preamble_tl
2700   {
2701     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2702     #1
2703     < \@@_cell_end:
2704   }

```

We test for the presence of a `<`.

```

2705   \@@_patch_m_preamble_x:n
2706 }

```

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

```

2707 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2708 {
2709   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2710   \@@_patch_m_preamble:n
2711 }

```

For `|`

```

2712 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2713 {
2714   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2715   \@@_patch_m_preamble:n
2716 }

```

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

```

2717 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2718 {
2719   \tl_gput_right:Nn \g_@@_preamble_tl
2720   {
2721     > {
2722       \@@_cell_begin:w
2723       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }

```

```

2724         \mode_leave_vertical:
2725         \arraybackslash
2726         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2727     }
2728     c
2729     < {
2730         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2731         \end { minipage }
2732         \@@_cell_end:
2733     }
2734 }

```

We test for the presence of a <.

```

2735     \@@_patch_m_preamble_x:n
2736 }

```

For w and W

```

2737 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2738 {
2739     \tl_gput_right:Nn \g_@@_preamble_tl
2740     {
2741         > {
2742             \dim_set:Nn \l_@@_col_width_dim { #4 }
2743             \hbox_set:Nw \l_@@_cell_box
2744             \@@_cell_begin:w
2745             \str_set:Nn \l_@@_hpos_cell_str { #3 }
2746         }
2747         c
2748         < {
2749             \@@_cell_end:
2750             \hbox_set_end:
2751             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2752             #1
2753             \@@_adjust_size_box:
2754             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2755         }
2756     }

```

We test for the presence of a <.

```

2757     \@@_patch_m_preamble_x:n
2758 }

```

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

```

2759 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2760 {
2761     \str_if_eq:nnTF { #1 } { < }
2762     \@@_patch_m_preamble_ix:n
2763     { \@@_patch_m_preamble:n { #1 } }
2764 }
2765 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2766 {
2767     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2768     \@@_patch_m_preamble_x:n
2769 }

```

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).

```

2770 \cs_new_protected:Npn \@@_put_box_in_flow:
2771 {
2772     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2773     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }

```

```

2774 \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2775 { \box_use_drop:N \l_tmpa_box }
2776 \@@_put_box_in_flow_i:
2777 }

```

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).

```

2778 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2779 {
2780 \pgfpicture
2781 \@@_qpoint:n { row - 1 }
2782 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2783 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2784 \dim_gadd:Nn \g_tmpa_dim \pgf@y
2785 \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).

```

2786 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2787 {
2788 \int_set:Nn \l_tmpa_int
2789 {
2790 \str_range:Nnn
2791 \l_@@_baseline_tl
2792 6
2793 { \tl_count:V \l_@@_baseline_tl }
2794 }
2795 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2796 }
2797 {
2798 \str_case:VnF \l_@@_baseline_tl
2799 {
2800 { t } { \int_set:Nn \l_tmpa_int 1 }
2801 { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2802 }
2803 { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2804 \bool_lazy_or:nnT
2805 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2806 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2807 {
2808 \@@_error:n { bad~value~for~baseline }
2809 \int_set:Nn \l_tmpa_int 1
2810 }
2811 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2812 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2813 }
2814 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2815 \endpgfpicture
2816 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2817 \box_use_drop:N \l_tmpa_box
2818 }

```

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).

```

2819 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2820 {

```


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.

```

2821 \bool_lazy_and:nNT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2822 {
2823   \box_set_wd:Nn \l_@@_the_array_box
2824   { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2825 }

```

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

```

2826 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2827 \bool_if:NT \l_@@_caption_above_bool
2828 {
2829   \tl_if_empty:NF \l_@@_caption_tl
2830   {
2831     \bool_set_false:N \g_@@_caption_finished_bool
2832     \int_gzero:N \c@tabularnote
2833     \@@_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).

```

2834 \int_compare:nNnT \g_@@_notes_caption_int > 0
2835 {
2836   \tl_gput_right:Nx \g_@@_aux_tl
2837   {
2838     \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2839     { \int_use:N \g_@@_notes_caption_int }
2840   }
2841   \int_gzero:N \g_@@_notes_caption_int
2842 }
2843 }
2844 }

```

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

```

2845 \hbox
2846 {
2847   \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.

```

2848 \@@_create_extra_nodes:
2849 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2850 }

```

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).

```

2851 \bool_lazy_any:nT
2852 {
2853   { ! \seq_if_empty_p:N \g_@@_notes_seq }
2854   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2855   { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2856 }
2857 \@@_insert_tabularnotes:
2858 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2859 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2860 \end { minipage }
2861 }

```

```

2862 \cs_new_protected:Npn \@@_insert_caption:
2863 {
2864   \tl_if_empty:NF \l_@@_caption_tl
2865   {
2866     \cs_if_exist:NTF \@capytype
2867     { \@@_insert_caption_i: }
2868     { \@@_error:n { caption-outside~float } }
2869   }
2870 }

```

```

2871 \cs_new_protected:Npn \@@_insert_caption_i:
2872 {
2873   \group_begin:

```

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

```

2874   \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.

```

2875   \IfPackageLoadedTF { floatrow }
2876   { \cs_set_eq:NN \@makecaption \FR@makecaption }
2877   { }
2878   \tl_if_empty:NTF \l_@@_short_caption_tl
2879   { \caption }
2880   { \caption [ \l_@@_short_caption_tl ] }
2881   { \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.

```

2882   \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
2883   {
2884     \bool_gset_true:N \g_@@_caption_finished_bool
2885     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
2886     \int_gzero:N \c@tabularnote
2887   }
2888   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2889   \group_end:
2890 }

```

```

2891 \cs_new_protected:Npn \@@_tabularnote_error:n #1
2892 {
2893   \@@_error_or_warning:n { tabularnote~below~the~tabular }
2894   \@@_gredirect_none:n { tabularnote~below~the~tabular }
2895 }

```

```

2896 \cs_new_protected:Npn \@@_insert_tabularnotes:
2897 {
2898   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2899   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2900   \skip_vertical:N 0.65ex

```

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

```

2901   \group_begin:
2902   \l_@@_notes_code_before_tl
2903   \tl_if_empty:NF \g_@@_tabularnote_tl
2904   {
2905     \g_@@_tabularnote_tl \par
2906     \tl_gclear:N \g_@@_tabularnote_tl
2907   }

```

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.

```

2908 \int_compare:nNnT \c@tabularnote > 0
2909 {
2910   \bool_if:NTF \l_@@_notes_para_bool
2911   {
2912     \begin { tabularnotes* }
2913     \seq_map_inline:Nn \g_@@_notes_seq
2914       { \@@_one_tabularnote:nn ##1 }
2915     \strut
2916     \end { tabularnotes* }

```

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

```

2917   \par
2918 }
2919 {
2920   \tabularnotes
2921   \seq_map_inline:Nn \g_@@_notes_seq
2922     { \@@_one_tabularnote:nn ##1 }
2923   \strut
2924   \endtabularnotes
2925 }
2926 }
2927 \unskip
2928 \group_end:
2929 \bool_if:NT \l_@@_notes_bottomrule_bool
2930 {
2931   \IfPackageLoadedTF { booktabs }
2932   {

```

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

```

2933   \skip_vertical:N \aboverulesep

```

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

```

2934   { \CT@arc@ \hrule height \heavyrulewidth }
2935 }
2936 { \@@_error_or_warning:n { bottomrule-without-booktabs } }
2937 }
2938 \l_@@_notes_code_after_tl
2939 \seq_gclear:N \g_@@_notes_seq
2940 \seq_gclear:N \g_@@_notes_in_caption_seq
2941 \int_gzero:N \c@tabularnote
2942 }

```

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.

```

2943 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
2944 {
2945   \tl_if_novalue:nTF { #1 }
2946   { \item }
2947   { \item [ \@@_notes_label_in_list:n { #1 } ] }
2948 }

```

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`.

```

2949 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2950 {
2951   \pgfpicture
2952   \@@_qpoint:n { row - 1 }
2953   \dim_gset_eq:NN \g_tmpa_dim \pgf@y

```

```

2954 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2955 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2956 \endpgfpicture
2957 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2958 \int_compare:nNnT \l_@@_first_row_int = 0
2959 {
2960 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2961 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2962 }
2963 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2964 }

```

Now, the general case.

```

2965 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2966 {

```

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

```

2967 \tl_if_eq:NnT \l_@@_baseline_tl { t }
2968 { \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 .

```

2969 \pgfpicture
2970 \@@_qpoint:n { row - 1 }
2971 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2972 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2973 {
2974 \int_set:Nn \l_tmpa_int
2975 {
2976 \str_range:Nnn
2977 \l_@@_baseline_tl
2978 6
2979 { \tl_count:V \l_@@_baseline_tl }
2980 }
2981 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2982 }
2983 {
2984 \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2985 \bool_lazy_or:nnT
2986 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2987 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2988 {
2989 \@@_error:n { bad-value-for-baseline }
2990 \int_set:Nn \l_tmpa_int 1
2991 }
2992 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2993 }
2994 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2995 \endpgfpicture
2996 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2997 \int_compare:nNnT \l_@@_first_row_int = 0
2998 {
2999 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3000 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3001 }
3002 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3003 }

```

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.

```

3004 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3005 {

```

We will compute the real width of both delimiters used.

```

3006 \dim_zero_new:N \l_@@_real_left_delim_dim
3007 \dim_zero_new:N \l_@@_real_right_delim_dim
3008 \hbox_set:Nn \l_tmpb_box
3009 {
3010   \c_math_toggle_token
3011   \left #1
3012   \vcenter
3013   {
3014     \vbox_to_ht:nn
3015       { \box_ht_plus_dp:N \l_tmpa_box }
3016       { }
3017   }
3018   \right .
3019   \c_math_toggle_token
3020 }
3021 \dim_set:Nn \l_@@_real_left_delim_dim
3022 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3023 \hbox_set:Nn \l_tmpb_box
3024 {
3025   \c_math_toggle_token
3026   \left .
3027   \vbox_to_ht:nn
3028     { \box_ht_plus_dp:N \l_tmpa_box }
3029     { }
3030   \right #2
3031   \c_math_toggle_token
3032 }
3033 \dim_set:Nn \l_@@_real_right_delim_dim
3034 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3035 \skip_horizontal:N \l_@@_left_delim_dim
3036 \skip_horizontal:N -\l_@@_real_left_delim_dim
3037 \@@_put_box_in_flow:
3038 \skip_horizontal:N \l_@@_right_delim_dim
3039 \skip_horizontal:N -\l_@@_real_right_delim_dim
3040 }

```

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).

```

3041 \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).

```

3042 {
3043   \peek_remove_spaces:n
3044   {
3045     \peek_meaning:NTF \end
3046     \@@_analyze_end:Nn
3047     {
3048       \@@_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`).

```

3049     \@@_array:V \g_@@_preamble_tl
3050   }
3051 }
3052 }

```

```

3053 {
3054   \@@_create_col_nodes:
3055   \endarray
3056 }

```

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

```

3057 \NewDocumentEnvironment { @@-light-syntax } { b }
3058 {

```

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`.

```

3059   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3060   \tl_map_inline:nn { #1 }
3061   {
3062     \str_if_eq:nnT { ##1 } { & }
3063     { \@@_fatal:n { ampersand-in-light-syntax } }
3064     \str_if_eq:nnT { ##1 } { \ }
3065     { \@@_fatal:n { double-backslash-in-light-syntax } }
3066   }

```

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`.

```

3067   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3068 }

```

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.

```

3069 {
3070   \@@_create_col_nodes:
3071   \endarray
3072 }

3073 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3074 {
3075   \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).

```

3076   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3077   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3078   \seq_set_split:NvN \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3079   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3080   \tl_if_empty:NF \l_tmpa_tl
3081   { \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.

```

3082   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3083   { \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`).

```
3084 \tl_clear_new:N \l_@@_new_body_tl
3085 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3086 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3087 \@@_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).

```
3088 \seq_map_inline:Nn \l_@@_rows_seq
3089 {
3090   \tl_put_right:Nn \l_@@_new_body_tl { \\ }
3091   \@@_line_with_light_syntax:n { #1 }
3092 }
3093 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3094 {
3095   \int_set:Nn \l_@@_last_col_int
3096   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3097 }
```

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.

```
3098 \@@_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`).

```
3099 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3100 }
3101 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3102 {
3103   \seq_clear_new:N \l_@@_cells_seq
3104   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3105   \int_set:Nn \l_@@_nb_cols_int
3106   {
3107     \int_max:nn
3108     \l_@@_nb_cols_int
3109     { \seq_count:N \l_@@_cells_seq }
3110   }
3111   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3112   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3113   \seq_map_inline:Nn \l_@@_cells_seq
3114   { \tl_put_right:Nn \l_@@_new_body_tl { & #1 } }
3115 }
3116 \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`.

```
3117 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3118 {
3119   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3120   { \@@_fatal:n { empty~environment } }
```

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

```
3121 \end { #2 }
3122 }
```

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).

```

3123 \cs_new:Npn \@@_create_col_nodes:
3124 {
3125   \crrc
3126   \int_compare:nNnT \l_@@_first_col_int = 0
3127   {
3128     \omit
3129     \hbox_overlap_left:n
3130     {
3131       \bool_if:NT \l_@@_code_before_bool
3132       { \pgfsys@markposition { \@@_env: - col - 0 } }
3133       \pgfpicture
3134       \pgfrememberpicturerepositiononpagetrue
3135       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3136       \str_if_empty:NF \l_@@_name_str
3137       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3138       \endpgfpicture
3139       \skip_horizontal:N 2\col@sep
3140       \skip_horizontal:N \g_@@_width_first_col_dim
3141     }
3142     &
3143   }
3144   \omit

```

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

```

3145   \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`).

```

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

```



```

3179     \str_if_empty:NF \l_@@_name_str
3180     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3181     \endpgfpicture
3182 }

```

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.

```

3183     \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3184     \bool_if:NF \l_@@_auto_columns_width_bool
3185     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3186     {
3187         \bool_lazy_and:nnTF
3188         \l_@@_auto_columns_width_bool
3189         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3190         { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3191         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3192         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3193     }
3194     \skip_horizontal:N \g_tmpa_skip
3195     \hbox
3196     {
3197         \bool_if:NT \l_@@_code_before_bool
3198         {
3199             \hbox
3200             {
3201                 \skip_horizontal:N -0.5\arrayrulewidth
3202                 \pgfsys@markposition { \@@_env: - col - 2 }
3203                 \skip_horizontal:N 0.5\arrayrulewidth
3204             }
3205         }
3206         \pgfpicture
3207         \pgfrememberpicturepositiononpagetrue
3208         \pgfcoordinate { \@@_env: - col - 2 }
3209         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3210         \str_if_empty:NF \l_@@_name_str
3211         { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3212         \endpgfpicture
3213     }

```

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.

```

3214     \int_gset:Nn \g_tmpa_int 1
3215     \bool_if:NTF \g_@@_last_col_found_bool
3216     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3217     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3218     {
3219         &
3220         \omit
3221         \int_gincr:N \g_tmpa_int

```

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

```

3222     \skip_horizontal:N \g_tmpa_skip
3223     \bool_if:NT \l_@@_code_before_bool
3224     {
3225         \hbox
3226         {
3227             \skip_horizontal:N -0.5\arrayrulewidth
3228             \pgfsys@markposition
3229             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3230             \skip_horizontal:N 0.5\arrayrulewidth

```

```

3231     }
3232   }

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

3233   \pgfpicture
3234     \pgfrememberpicturepositiononpagetrue
3235     \pgfcoordinate { \l_@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3236     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3237     \str_if_empty:NF \l_@@_name_str
3238     {
3239       \pgfnodealias
3240       { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3241       { \l_@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3242     }
3243   \endpgfpicture
3244 }

3245   &
3246   \omit

```

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

```

3247   \int_compare:nNnT \g_@@_col_total_int = 1
3248   { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3249   \skip_horizontal:N \g_tmpa_skip
3250   \int_gincr:N \g_tmpa_int
3251   \bool_lazy_all:nT
3252   {
3253     \g_@@_NiceArray_bool
3254     { \bool_not_p:n \l_@@_NiceTabular_bool }
3255     { \clist_if_empty_p:N \l_@@_vlines_clist }
3256     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3257     { ! \l_@@_bar_at_end_of_pream_bool }
3258   }
3259   { \skip_horizontal:N -\col@sep }
3260   \bool_if:NT \l_@@_code_before_bool
3261   {
3262     \hbox
3263     {
3264       \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.

```

3265       \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3266       { \skip_horizontal:N -\arraycolsep }
3267       \pgfsys@markposition
3268       { \l_@@_env: - col - \int_eval:n {
3269         \g_tmpa_int + 1 } }
3270       \skip_horizontal:N 0.5\arrayrulewidth
3271       \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3272       { \skip_horizontal:N \arraycolsep }
3273     }
3274   }
3275   \pgfpicture
3276     \pgfrememberpicturepositiononpagetrue
3277     \pgfcoordinate { \l_@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3278     {
3279       \bool_lazy_and:nnTF \l_@@_Matrix_bool \g_@@_NiceArray_bool
3280       {
3281         \pgfpoint
3282         { - 0.5 \arrayrulewidth - \arraycolsep }
3283         \c_zero_dim
3284       }

```

```

3285         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3286     }
3287     \str_if_empty:NF \l_@@_name_str
3288     {
3289         \pgfnodealias
3290         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3291         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3292     }
3293     \endpgfpicture

3294     \bool_if:NT \g_@@_last_col_found_bool
3295     {
3296         \hbox_overlap_right:n
3297         {
3298             \skip_horizontal:N \g_@@_width_last_col_dim
3299             \bool_if:NT \l_@@_code_before_bool
3300             {
3301                 \pgfsys@markposition
3302                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3303             }
3304             \pgfpicture
3305             \pgfrememberpicturerepositiononpagetrue
3306             \pgfcoordinate
3307             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3308             \pgfpointorigin
3309             \str_if_empty:NF \l_@@_name_str
3310             {
3311                 \pgfnodealias
3312                 {
3313                     \l_@@_name_str - col
3314                     - \int_eval:n { \g_@@_col_total_int + 1 }
3315                 }
3316                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3317             }
3318             \endpgfpicture
3319         }
3320     }
3321     \cr
3322 }

```

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

```

3323 \tl_const:Nn \c_@@_preamble_first_col_tl
3324 {
3325     >
3326     {

```

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).

```

3327     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3328     \bool_gset_true:N \g_@@_after_col_zero_bool
3329     \@@_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.

```

3330     \hbox_set:Nw \l_@@_cell_box
3331     \@@_math_toggle_token:
3332     \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”.

```

3333     \bool_lazy_and:nnT
3334     { \int_compare_p:nNn \c@iRow > 0 }

```

```

3335     {
3336         \bool_lazy_or_p:nn
3337         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3338         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3339     }
3340     {
3341         \l_@@_code_for_first_col_tl
3342         \xglobal \colorlet { nicematrix-first-col } { . }
3343     }
3344 }

```

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`.

```

3345     l
3346     <
3347     {
3348         \@@_math_toggle_token:
3349         \hbox_set_end:
3350         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3351         \@@_adjust_size_box:
3352         \@@_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.

```

3353         \dim_gset:Nn \g_@@_width_first_col_dim
3354         { \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.

```

3355         \hbox_overlap_left:n
3356         {
3357             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3358             \@@_node_for_cell:
3359             { \box_use_drop:N \l_@@_cell_box }
3360             \skip_horizontal:N \l_@@_left_delim_dim
3361             \skip_horizontal:N \l_@@_left_margin_dim
3362             \skip_horizontal:N \l_@@_extra_left_margin_dim
3363         }
3364         \bool_gset_false:N \g_@@_empty_cell_bool
3365         \skip_horizontal:N -2\col@sep
3366     }
3367 }

```

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

```

3368 \tl_const:Nn \c_@@_preamble_last_col_tl
3369 {
3370     >
3371     {

```

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

```

3372         \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.

```

3373         \bool_gset_true:N \g_@@_last_col_found_bool
3374         \int_gincr:N \c@jCol
3375         \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.

```

3376         \hbox_set:Nw \l_@@_cell_box
3377         \@@_math_toggle_token:
3378         \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”.

```

3379     \int_compare:nNnT \c@iRow > 0
3380     {
3381         \bool_lazy_or:nnT
3382         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3383         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3384         {
3385             \l_@@_code_for_last_col_tl
3386             \xglobal \colorlet { nicematrix-last-col } { . }
3387         }
3388     }
3389 }
3390 1
3391 <
3392 {
3393     \@@_math_toggle_token:
3394     \hbox_set_end:
3395     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3396     \@@_adjust_size_box:
3397     \@@_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.

```

3398     \dim_gset:Nn \g_@@_width_last_col_dim
3399     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3400     \skip_horizontal:N -2\col@sep

```

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

```

3401     \hbox_overlap_right:n
3402     {
3403         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3404         {
3405             \skip_horizontal:N \l_@@_right_delim_dim
3406             \skip_horizontal:N \l_@@_right_margin_dim
3407             \skip_horizontal:N \l_@@_extra_right_margin_dim
3408             \@@_node_for_cell:
3409         }
3410     }
3411     \bool_gset_false:N \g_@@_empty_cell_bool
3412 }
3413 }

```

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

```

3414 \NewDocumentEnvironment { NiceArray } { }
3415 {
3416     \bool_gset_true:N \g_@@_NiceArray_bool
3417     \str_if_empty:NT \g_@@_name_env_str
3418     { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

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

```

3419     \NiceArrayWithDelims . .
3420 }
3421 { \endNiceArrayWithDelims }

```

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

```

3422 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3423 {

```

```

3424 \NewDocumentEnvironment { #1 NiceArray } { }
3425 {
3426   \bool_gset_false:N \g_@@_NiceArray_bool
3427   \str_if_empty:NT \g_@@_name_env_str
3428   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3429   \@@_test_if_math_mode:
3430   \NiceArrayWithDelims #2 #3
3431 }
3432 { \endNiceArrayWithDelims }
3433 }
3434 \@@_def_env:nnn p ( )
3435 \@@_def_env:nnn b [ ]
3436 \@@_def_env:nnn B \{ \}
3437 \@@_def_env:nnn v | |
3438 \@@_def_env:nnn V \| \|

```

13 The environment {NiceMatrix} and its variants

```

3439 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3440 {
3441   \bool_set_true:N \l_@@_Matrix_bool
3442   \use:c { #1 NiceArray }
3443   {
3444     *
3445     {
3446       \int_case:nnF \l_@@_last_col_int
3447       {
3448         { -2 } { \c@MaxMatrixCols }
3449         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3450       }
3451       { \int_eval:n { \l_@@_last_col_int - 1 } }
3452     }
3453     { #2 }
3454   }
3455 }
3456 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3457 \clist_map_inline:nn { p , b , B , v , V }
3458 {
3459   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3460   {
3461     \bool_gset_false:N \g_@@_NiceArray_bool
3462     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3463     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3464     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3465   }
3466   { \use:c { end #1 NiceArray } }
3467 }

```

We define also an environment {NiceMatrix}

```

3468 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3469 {
3470   \bool_gset_false:N \g_@@_NiceArray_bool
3471   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3472   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3473   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3474 }

```

```
3475 { \endNiceArray }
```

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

```
3476 \cs_new_protected:Npn \@@_NotEmpty:
3477 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3478 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3479 {
```

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.

```
3480   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3481     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3482   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3483   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3484   \int_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
3485     {
3486       \bool_if:NT \l_@@_hvlines_bool
3487       {
3488         \bool_set_true:N \l_@@_except_borders_bool
3489         % we should try to be more efficient in the number of lines of code here
3490         \tl_if_empty:NTF \l_@@_rules_color_tl
3491           {
3492             \tl_gput_right:Nn \g_@@_pre_code_after_tl
3493               {
3494                 \@@_stroke_block:nnn
3495                   { rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim }
3496                   { 1-1 }
3497                   { \int_use:N \c@iRow - \int_use:N \c@jCol }
3498               }
3499             }
3500           {
3501             \tl_gput_right:Nn \g_@@_pre_code_after_tl
3502               {
3503                 \@@_stroke_block:nnn
3504                   {
3505                     rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3506                     draw = \l_@@_rules_color_tl
3507                   }
3508                   { 1-1 }
3509                   { \int_use:N \c@iRow - \int_use:N \c@jCol }
3510               }
3511             }
3512           }
3513         }
3514       \tl_if_empty:NF \l_@@_short_caption_tl
3515       {
3516         \tl_if_empty:NT \l_@@_caption_tl
3517         {
3518           \@@_error_or_warning:n { short-caption~without~caption }
3519           \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3520         }
3521       }
3522       \tl_if_empty:NF \l_@@_label_tl
3523       {
3524         \tl_if_empty:NT \l_@@_caption_tl
3525         { \@@_error_or_warning:n { label~without~caption } }
3526       }
3527     \NewDocumentEnvironment { TabularNote } { b }
3528     {
```

```

3529 \bool_if:NTF \l_@@_in_code_after_bool
3530 { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3531 {
3532   \tl_if_empty:NF \g_@@_tabularnote_tl
3533   { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3534   \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3535 }
3536 }
3537 { }
3538 \bool_set_true:N \l_@@_NiceTabular_bool
3539 \NiceArray { #2 }
3540 }
3541 { \endNiceArray }

3542 \cs_set_protected:Npn \@@_newcolumnntype #1
3543 {
3544   \cs_if_free:cT { NC @ find @ #1 }
3545   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3546   \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3547   \peek_meaning:NTF [
3548     { \newcol@ #1 }
3549     { \newcol@ #1 [ 0 ] }
3550   }

```

```

3551 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3552 {

```

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}`).

```

3553 \IfPackageLoadedTF { tabularx }
3554 { \newcolumnntype { X } { \@@_X } }
3555 { }
3556 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3557 \dim_zero_new:N \l_@@_width_dim
3558 \dim_set:Nn \l_@@_width_dim { #1 }
3559 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3560 \bool_set_true:N \l_@@_NiceTabular_bool
3561 \NiceArray { #3 }
3562 }
3563 { \endNiceArray }

3564 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
3565 {
3566   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3567   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3568   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3569   \bool_set_true:N \l_@@_NiceTabular_bool
3570   \NiceArray { #3 }
3571 }
3572 { \endNiceArray }

```

15 After the construction of the array

```

3573 \cs_new_protected:Npn \@@_after_array:
3574 {
3575   \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.

```
3576 \bool_if:NT \g_@@_last_col_found_bool
3577 { \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.

```
3578 \bool_if:NT \l_@@_last_col_without_value_bool
3579 { \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.

```
3580 \bool_if:NT \l_@@_last_row_without_value_bool
3581 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3582 \tl_gput_right:Nx \g_@@_aux_tl
3583 {
3584   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3585   {
3586     \int_use:N \l_@@_first_row_int ,
3587     \int_use:N \c@iRow ,
3588     \int_use:N \g_@@_row_total_int ,
3589     \int_use:N \l_@@_first_col_int ,
3590     \int_use:N \c@jCol ,
3591     \int_use:N \g_@@_col_total_int
3592   }
3593 }
```

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).

```
3594 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3595 {
3596   \tl_gput_right:Nx \g_@@_aux_tl
3597   {
3598     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3599     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3600   }
3601 }
3602 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3603 {
3604   \tl_gput_right:Nx \g_@@_aux_tl
3605   {
3606     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3607     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3608     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3609     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3610   }
3611 }
```

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

```
3612 \@@_create_diag_nodes:
```

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

```
3613 \pgfpicture
3614 \int_step_inline:nn \c@iRow
3615 {
3616   \pgfnodelalias
3617   { \@@_env: - ##1 - last }
3618   { \@@_env: - ##1 - \int_use:N \c@jCol }
3619 }
3620 \int_step_inline:nn \c@jCol
3621 {
3622   \pgfnodelalias
3623   { \@@_env: - last - ##1 }
3624   { \@@_env: - \int_use:N \c@iRow - ##1 }
```

```

3625     }
3626     \str_if_empty:NF \l_@@_name_str
3627     {
3628         \int_step_inline:nn \c@iRow
3629         {
3630             \pgfnodealias
3631             { \l_@@_name_str - ##1 - last }
3632             { \@@_env: - ##1 - \int_use:N \c@jCol }
3633         }
3634         \int_step_inline:nn \c@jCol
3635         {
3636             \pgfnodealias
3637             { \l_@@_name_str - last - ##1 }
3638             { \@@_env: - \int_use:N \c@iRow - ##1 }
3639         }
3640     }
3641     \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.

```

3642     \bool_if:NT \l_@@_parallelize_diags_bool
3643     {
3644         \int_gzero_new:N \g_@@_ddots_int
3645         \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.

```

3646         \dim_gzero_new:N \g_@@_delta_x_one_dim
3647         \dim_gzero_new:N \g_@@_delta_y_one_dim
3648         \dim_gzero_new:N \g_@@_delta_x_two_dim
3649         \dim_gzero_new:N \g_@@_delta_y_two_dim
3650     }
3651     \int_zero_new:N \l_@@_initial_i_int
3652     \int_zero_new:N \l_@@_initial_j_int
3653     \int_zero_new:N \l_@@_final_i_int
3654     \int_zero_new:N \l_@@_final_j_int
3655     \bool_set_false:N \l_@@_initial_open_bool
3656     \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.

```

3657     \bool_if:NT \l_@@_small_bool
3658     {
3659         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3660         \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.

```

3661         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3662         { 0.6 \l_@@_xdots_shorten_start_dim }
3663         \dim_set:Nn \l_@@_xdots_shorten_end_dim
3664         { 0.6 \l_@@_xdots_shorten_end_dim }
3665     }

```

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

```

3666     \@@_draw_dotted_lines:

```

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

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.

```
3667 \@@_compute_corners:
```

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

```
3668 \@@_adjust_pos_of_blocks_seq:
3669 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3670 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3671 \IfPackageLoadedTF { tikz }
3672 {
3673   \tikzset
3674   {
3675     every~picture / .style =
3676     {
3677       overlay ,
3678       remember~picture ,
3679       name~prefix = \@@_env: -
3680     }
3681   }
3682   { }
3683   \cs_set_eq:NN \ialign \@@_old_ialign:
3684   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3685   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3686   \cs_set_eq:NN \OverBrace \@@_OverBrace
3687   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3688   \cs_set_eq:NN \line \@@_line
3689   \g_@@_pre_code_after_tl
3690   \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.

```
3692 \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).

```
3693 \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`).

```
3694 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3695 { \@@_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:`.

```
3696 \bool_set_true:N \l_@@_in_code_after_bool
3697 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3698 \scan_stop:
3699 \tl_gclear:N \g_nicematrix_code_after_tl
3700 \group_end:
```

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

```
3701 \tl_if_empty:NF \g_@@_pre_code_before_tl
3702 {
```

```

3703     \tl_gput_right:Nx \g_@@_aux_tl
3704     {
3705         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3706         { \exp_not:V \g_@@_pre_code_before_tl }
3707     }
3708     \tl_gclear:N \g_@@_pre_code_before_tl
3709 }
3710 \tl_if_empty:NF \g_nicematrix_code_before_tl
3711 {
3712     \tl_gput_right:Nx \g_@@_aux_tl
3713     {
3714         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3715         { \exp_not:V \g_nicematrix_code_before_tl }
3716     }
3717     \tl_gclear:N \g_nicematrix_code_before_tl
3718 }

3719 \str_gclear:N \g_@@_name_env_str
3720 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹². This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

3721     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3722 }

```

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`.

```

3723 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3724 { \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).

```

3725 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3726 {
3727     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3728     { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3729 }

```

The following command must *not* be protected.

```

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

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3743 }
3744 { #5 }
3745 }

```

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).

```

3746 \hook_gput_code:nnn { begindocument } { . }
3747 {
3748   \cs_new_protected:Npx \@@_draw_dotted_lines:
3749   {
3750     \c_@@_pgfortikzpicture_tl
3751     \@@_draw_dotted_lines_i:
3752     \c_@@_endpgfortikzpicture_tl
3753   }
3754 }

```

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

```

3755 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3756 {
3757   \pgfrememberpicturepositiononpagetrue
3758   \pgf@relevantforpicturesizefalse
3759   \g_@@_HVdotsfor_lines_tl
3760   \g_@@_Vdots_lines_tl
3761   \g_@@_Ddots_lines_tl
3762   \g_@@_Iddots_lines_tl
3763   \g_@@_Cdots_lines_tl
3764   \g_@@_Ldots_lines_tl
3765 }

3766 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3767 {
3768   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3769   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3770 }

```

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

```

3771 \pgfdeclareshape { @@_diag_node }
3772 {
3773   \savedanchor { \five }
3774   {
3775     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3776     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3777   }
3778   \anchor { 5 } { \five }
3779   \anchor { center } { \pgfpointorigin }
3780 }

```

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).

```

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

```

```

3791 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3792 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3793 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3794 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3795 \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 `@ä_diag_node`) that we will construct.

```

3796 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3797 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3798 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3799 \str_if_empty:NF \l_@@_name_str
3800 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3801 }

```

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

```

3802 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3803 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3804 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3805 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3806 \pgfcoordinate
3807 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3808 \pgfnodealias
3809 { \@@_env: - last }
3810 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3811 \str_if_empty:NF \l_@@_name_str
3812 {
3813   \pgfnodealias
3814   { \l_@@_name_str - \int_use:N \l_tmpa_int }
3815   { \@@_env: - \int_use:N \l_tmpa_int }
3816   \pgfnodealias
3817   { \l_@@_name_str - last }
3818   { \@@_env: - last }
3819 }
3820 \endpgfpicture
3821 }

```

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.

```
3822 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3823 {
```

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

```
3824 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3825 \int_set:Nn \l_@@_initial_i_int { #1 }
3826 \int_set:Nn \l_@@_initial_j_int { #2 }
3827 \int_set:Nn \l_@@_final_i_int { #1 }
3828 \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.

```
3829 \bool_set_false:N \l_@@_stop_loop_bool
3830 \bool_do_until:Nn \l_@@_stop_loop_bool
3831 {
3832   \int_add:Nn \l_@@_final_i_int { #3 }
3833   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3834   \bool_set_false:N \l_@@_final_open_bool
3835   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3836   {
3837     \int_compare:nNnTF { #3 } = 1
3838     { \bool_set_true:N \l_@@_final_open_bool }
3839     {
3840       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3841       { \bool_set_true:N \l_@@_final_open_bool }
3842     }
3843   }
3844   {
3845     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3846     {
3847       \int_compare:nNnT { #4 } = { -1 }
3848       { \bool_set_true:N \l_@@_final_open_bool }
3849     }
3850     {
3851       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3852       {
3853         \int_compare:nNnT { #4 } = 1
3854         { \bool_set_true:N \l_@@_final_open_bool }
3855       }
3856     }
3857   }
3858   \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.

```
3859   {
```

We do a step backwards.

```
3860     \int_sub:Nn \l_@@_final_i_int { #3 }
3861     \int_sub:Nn \l_@@_final_j_int { #4 }
3862     \bool_set_true:N \l_@@_stop_loop_bool
3863   }
```

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`.

```

3864     {
3865         \cs_if_exist:cTF
3866         {
3867             @@ _ dotted _
3868             \int_use:N \l_@@_final_i_int -
3869             \int_use:N \l_@@_final_j_int
3870         }
3871         {
3872             \int_sub:Nn \l_@@_final_i_int { #3 }
3873             \int_sub:Nn \l_@@_final_j_int { #4 }
3874             \bool_set_true:N \l_@@_final_open_bool
3875             \bool_set_true:N \l_@@_stop_loop_bool
3876         }
3877         {
3878             \cs_if_exist:cTF
3879             {
3880                 pgf @ sh @ ns @ \@@_env:
3881                 - \int_use:N \l_@@_final_i_int
3882                 - \int_use:N \l_@@_final_j_int
3883             }
3884             { \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.

```

3885         {
3886             \cs_set:cpn
3887             {
3888                 @@ _ dotted _
3889                 \int_use:N \l_@@_final_i_int -
3890                 \int_use:N \l_@@_final_j_int
3891             }
3892             { }
3893         }
3894     }
3895 }
3896

```

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

```

3897     \bool_set_false:N \l_@@_stop_loop_bool
3898     \bool_do_until:Nn \l_@@_stop_loop_bool
3899     {
3900         \int_sub:Nn \l_@@_initial_i_int { #3 }
3901         \int_sub:Nn \l_@@_initial_j_int { #4 }
3902         \bool_set_false:N \l_@@_initial_open_bool
3903         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3904         {
3905             \int_compare:nNnTF { #3 } = 1
3906             { \bool_set_true:N \l_@@_initial_open_bool }
3907             {
3908                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3909                 { \bool_set_true:N \l_@@_initial_open_bool }
3910             }
3911         }
3912         {
3913             \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3914             {

```



```

3915         \int_compare:nNtT { #4 } = 1
3916         { \bool_set_true:N \l_@@_initial_open_bool }
3917     }
3918     {
3919         \int_compare:nNtT \l_@@_initial_j_int > \l_@@_col_max_int
3920         {
3921             \int_compare:nNtT { #4 } = { -1 }
3922             { \bool_set_true:N \l_@@_initial_open_bool }
3923         }
3924     }
3925 }
3926 \bool_if:NTF \l_@@_initial_open_bool
3927 {
3928     \int_add:Nn \l_@@_initial_i_int { #3 }
3929     \int_add:Nn \l_@@_initial_j_int { #4 }
3930     \bool_set_true:N \l_@@_stop_loop_bool
3931 }
3932 {
3933     \cs_if_exist:cTF
3934     {
3935         @@ _ dotted _
3936         \int_use:N \l_@@_initial_i_int -
3937         \int_use:N \l_@@_initial_j_int
3938     }
3939     {
3940         \int_add:Nn \l_@@_initial_i_int { #3 }
3941         \int_add:Nn \l_@@_initial_j_int { #4 }
3942         \bool_set_true:N \l_@@_initial_open_bool
3943         \bool_set_true:N \l_@@_stop_loop_bool
3944     }
3945     {
3946         \cs_if_exist:cTF
3947         {
3948             pgf @ sh @ ns @ \@@_env:
3949             - \int_use:N \l_@@_initial_i_int
3950             - \int_use:N \l_@@_initial_j_int
3951         }
3952         { \bool_set_true:N \l_@@_stop_loop_bool }
3953     }
3954     \cs_set:cpn
3955     {
3956         @@ _ dotted _
3957         \int_use:N \l_@@_initial_i_int -
3958         \int_use:N \l_@@_initial_j_int
3959     }
3960     { }
3961 }
3962 }
3963 }
3964 }

```

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.

```

3965     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3966     {
3967         { \int_use:N \l_@@_initial_i_int }

```

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

```

3968     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3969     { \int_use:N \l_@@_final_i_int }
3970     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3971     { } % for the name of the block
3972 }

```

```
3973 }
```

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).

```
3974 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3975 {
3976   \int_set:Nn \l_@@_row_min_int 1
3977   \int_set:Nn \l_@@_col_min_int 1
3978   \int_set_eq:NN \l_@@_row_max_int \c{iRow}
3979   \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`.

```
3980   \seq_map_inline:Nn \g_@@_submatrix_seq
3981   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3982 }
```

#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.

```
3983 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3984 {
3985   \bool_if:nT
3986   {
3987     \int_compare_p:n { #3 <= #1 }
3988     && \int_compare_p:n { #1 <= #5 }
3989     && \int_compare_p:n { #4 <= #2 }
3990     && \int_compare_p:n { #2 <= #6 }
3991   }
3992   {
3993     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3994     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3995     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3996     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3997   }
3998 }
```

```
3999 \cs_new_protected:Npn \@@_set_initial_coords:
4000 {
4001   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4002   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4003 }
4004 \cs_new_protected:Npn \@@_set_final_coords:
4005 {
4006   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4007   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4008 }
4009 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4010 {
4011   \pgfpointanchor
4012   {
4013     \@@_env:
4014     - \int_use:N \l_@@_initial_i_int
4015     - \int_use:N \l_@@_initial_j_int
4016   }
4017   { #1 }
4018   \@@_set_initial_coords:
4019 }
4020 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4021 {
4022   \pgfpointanchor
```

```

4023 {
4024   \@@_env:
4025   - \int_use:N \l_@@_final_i_int
4026   - \int_use:N \l_@@_final_j_int
4027 }
4028 { #1 }
4029 \@@_set_final_coords:
4030 }
4031 \cs_new_protected:Npn \@@_open_x_initial_dim:
4032 {
4033   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4034   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4035   {
4036     \cs_if_exist:cT
4037     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4038     {
4039       \pgfpointanchor
4040       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4041       { west }
4042       \dim_set:Nn \l_@@_x_initial_dim
4043       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4044     }
4045   }

```

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

```

4046   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4047   {
4048     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4049     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4050     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4051   }
4052 }
4053 \cs_new_protected:Npn \@@_open_x_final_dim:
4054 {
4055   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4056   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4057   {
4058     \cs_if_exist:cT
4059     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4060     {
4061       \pgfpointanchor
4062       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4063       { east }
4064       \dim_set:Nn \l_@@_x_final_dim
4065       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4066     }
4067   }

```

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

```

4068   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4069   {
4070     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4071     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4072     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4073   }
4074 }

```

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.

```

4075 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4076 {
4077   \@@_adjust_to_submatrix:nn { #1 } { #2 }

```

```

4078 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4079 {
4080     \@@_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.

```

4081     \group_begin:
4082     \int_compare:nNnTF { #1 } = 0
4083     { \color { nicematrix-first-row } }
4084     {

```

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.

```

4085         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4086         { \color { nicematrix-last-row } }
4087     }
4088     \keys_set:nn { NiceMatrix / xdots } { #3 }
4089     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4090     \@@_actually_draw_Ldots:
4091 \group_end:
4092 }
4093 }

```

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`.

```

4094 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4095 {
4096     \bool_if:NTF \l_@@_initial_open_bool
4097     {
4098         \@@_open_x_initial_dim:
4099         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4100         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4101     }
4102     { \@@_set_initial_coords_from_anchor:n { base~east } }
4103     \bool_if:NTF \l_@@_final_open_bool
4104     {
4105         \@@_open_x_final_dim:
4106         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4107         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4108     }
4109     { \@@_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 (?).

```

4110     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4111     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4112     \@@_draw_line:
4113 }

```

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.

```

4114 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4115 {
4116   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4117   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4118   {
4119     \@@_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.

```

4120     \group_begin:
4121     \int_compare:nNnTF { #1 } = 0
4122     { \color { nicematrix-first-row } }
4123     {

```

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.

```

4124         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4125         { \color { nicematrix-last-row } }
4126     }
4127     \keys_set:nn { NiceMatrix / xdots } { #3 }
4128     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4129     \@@_actually_draw_Cdots:
4130     \group_end:
4131   }
4132 }

```

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`.

```

4133 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4134 {
4135   \bool_if:NTF \l_@@_initial_open_bool
4136   { \@@_open_x_initial_dim: }
4137   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4138   \bool_if:NTF \l_@@_final_open_bool
4139   { \@@_open_x_final_dim: }
4140   { \@@_set_final_coords_from_anchor:n { mid-west } }
4141   \bool_lazy_and:nnTF
4142   \l_@@_initial_open_bool
4143   \l_@@_final_open_bool
4144   {
4145     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4146     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4147     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4148     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4149     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4150   }
4151   {
4152     \bool_if:NT \l_@@_initial_open_bool
4153     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4154     \bool_if:NT \l_@@_final_open_bool
4155     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4156   }
4157   \@@_draw_line:
4158 }

```

```

4159 \cs_new_protected:Npn \@@_open_y_initial_dim:
4160 {
4161   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4162   \dim_set:Nn \l_@@_y_initial_dim
4163   {
4164     \fp_to_dim:n
4165     {
4166       \pgf@y
4167       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4168     }
4169   } % modified 6.13c
4170   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4171   {
4172     \cs_if_exist:cT
4173     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4174     {
4175       \pgfpointanchor
4176       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4177       { north }
4178       \dim_set:Nn \l_@@_y_initial_dim
4179       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4180     }
4181   }
4182 }
4183 \cs_new_protected:Npn \@@_open_y_final_dim:
4184 {
4185   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4186   \dim_set:Nn \l_@@_y_final_dim
4187   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4188   % modified 6.13c
4189   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4190   {
4191     \cs_if_exist:cT
4192     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4193     {
4194       \pgfpointanchor
4195       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4196       { south }
4197       \dim_set:Nn \l_@@_y_final_dim
4198       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4199     }
4200   }
4201 }

```

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.

```

4202 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4203 {
4204   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4205   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4206   {
4207     \@@_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.

```

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

```

```

4217         { \color { \l_@@_xdots_color_tl } }
4218     \@@_actually_draw_Vdots:
4219 \group_end:
4220 }
4221 }

```

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`.

```

4222 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4223 {

```

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

```

4224     \bool_set_false:N \l_tmpa_bool

```

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

```

4225     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
4226     {
4227         \@@_set_initial_coords_from_anchor:n { south-west }
4228         \@@_set_final_coords_from_anchor:n { north-west }
4229         \bool_set:Nn \l_tmpa_bool
4230         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4231     }

```

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

```

4232     \bool_if:NTF \l_@@_initial_open_bool
4233     \@@_open_y_initial_dim:
4234     { \@@_set_initial_coords_from_anchor:n { south } }
4235     \bool_if:NTF \l_@@_final_open_bool
4236     \@@_open_y_final_dim:
4237     { \@@_set_final_coords_from_anchor:n { north } }
4238     \bool_if:NTF \l_@@_initial_open_bool
4239     {
4240         \bool_if:NTF \l_@@_final_open_bool
4241         {
4242             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4243             \dim_set_eq:NN \l_tmpa_dim \pgf@x
4244             \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4245             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4246             \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.

```

4247     \int_compare:nNnT \l_@@_last_col_int > { -2 }
4248     {
4249         \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
4250         {
4251             \dim_set:Nn \l_tmpa_dim
4252             {
4253                 \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim
4254                 + 2 mm
4255             }
4256             \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim

```

```

4257         \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
4258     }
4259 }
4260 }
4261 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
4262 }
4263 {
4264     \bool_if:NTF \l_@@_final_open_bool
4265     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
4266     {

```

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.

```

4267         \dim_compare:nNf \l_@@_x_initial_dim = \l_@@_x_final_dim
4268         {
4269             \dim_set:Nn \l_@@_x_initial_dim
4270             {
4271                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4272                 \l_@@_x_initial_dim \l_@@_x_final_dim
4273             }
4274             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4275         }
4276     }
4277 }
4278 \@@_draw_line:
4279 }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4280 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4281 {
4282     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4283     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4284     {
4285         \@@_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.

```

4286     \group_begin:
4287     \keys_set:nn { NiceMatrix / xdots } { #3 }
4288     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4289     \@@_actually_draw_Ddots:
4290     \group_end:
4291 }
4292 }

```

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`.


```

4293 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4294 {
4295   \bool_if:NTF \l_@@_initial_open_bool
4296   {
4297     \@@_open_y_initial_dim:
4298     \@@_open_x_initial_dim:
4299   }
4300   { \@@_set_initial_coords_from_anchor:n { south-east } }
4301   \bool_if:NTF \l_@@_final_open_bool
4302   {
4303     \@@_open_x_final_dim:
4304     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4305   }
4306   { \@@_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.

```

4307   \bool_if:NT \l_@@_parallelize_diags_bool
4308   {
4309     \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).

```

4310     \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.

```

4311     {
4312       \dim_gset:Nn \g_@@_delta_x_one_dim
4313       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4314       \dim_gset:Nn \g_@@_delta_y_one_dim
4315       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4316     }

```

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`.

```

4317     {
4318       \dim_set:Nn \l_@@_y_final_dim
4319       {
4320         \l_@@_y_initial_dim +
4321         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4322         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4323       }
4324     }
4325   }
4326   \@@_draw_line:
4327 }

```

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.

```

4328 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4329 {
4330   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4331   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4332   {
4333     \@@_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.

```

4334   \group_begin:
4335   \keys_set:nn { NiceMatrix / xdots } { #3 }
4336   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4337   \@@_actually_draw_Iddots:

```

```

4338     \group_end:
4339   }
4340 }

```

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.`

```

4341 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4342 {
4343   \bool_if:NTF \l_@@_initial_open_bool
4344   {
4345     \@@_open_y_initial_dim:
4346     \@@_open_x_initial_dim:
4347   }
4348   { \@@_set_initial_coords_from_anchor:n { south-west } }
4349   \bool_if:NTF \l_@@_final_open_bool
4350   {
4351     \@@_open_y_final_dim:
4352     \@@_open_x_final_dim:
4353   }
4354   { \@@_set_final_coords_from_anchor:n { north-east } }
4355   \bool_if:NT \l_@@_parallelize_diags_bool
4356   {
4357     \int_gincr:N \g_@@_iddots_int
4358     \int_compare:nNnTF \g_@@_iddots_int = 1
4359     {
4360       \dim_gset:Nn \g_@@_delta_x_two_dim
4361       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4362       \dim_gset:Nn \g_@@_delta_y_two_dim
4363       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4364     }
4365     {
4366       \dim_set:Nn \l_@@_y_final_dim
4367       {
4368         \l_@@_y_initial_dim +
4369         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4370         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4371       }
4372     }
4373   }
4374   \@@_draw_line:
4375 }

```

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

```

4376 \cs_new_protected:Npn \@@_draw_line:
4377 {
4378   \pgfrememberpicturepositiononpagetrue
4379   \pgf@relevantforpicturesizefalse
4380   \bool_lazy_or:nnTF
4381     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4382     \l_@@_dotted_bool
4383     \@@_draw_standard_dotted_line:
4384     \@@_draw_unstandard_dotted_line:
4385 }

```

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.

```

4386 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4387 {
4388   \begin { scope }
4389   \@@_draw_unstandard_dotted_line:o
4390     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4391 }

```

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.

```

4392 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4393 {
4394   \@@_draw_unstandard_dotted_line:nVV
4395     { #1 }
4396     \l_@@_xdots_up_tl
4397     \l_@@_xdots_down_tl
4398 }
4399 \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.

```

4400 \hook_gput_code:nnn { begindocument } { . }
4401 {
4402   \IfPackageLoadedTF { tikz }
4403   {
4404     \tikzset
4405     {
4406       @@_node_above / .style = { sloped , above } ,
4407       @@_node_below / .style = { sloped , below }
4408     }
4409   }
4410   { }
4411 }

4412 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
4413 {
4414   \bool_if:NT \l_@@_xdots_h_labels_bool
4415   {
4416     \tikzset
4417     {

```

```

4418         @@_node_above / .style = { auto = left } ,
4419         @@_node_below / .style = { auto = right }
4420     }
4421 }
4422 \draw
4423 [
4424     shorten-> = \l_@@_xdots_shorten_end_dim ,
4425     shorten-< = \l_@@_xdots_shorten_start_dim ,
4426     #1
4427 ]
4428 ( \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).

```

4429     -- node [ @@_node_above ] { $ \scriptstyle #2 $ }
4430     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4431     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4432 \end { scope }
4433 }
4434 \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).

```

4435 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4436 {
4437     \bool_lazy_and:nnF
4438     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4439     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4440     {
4441         \pgfscope
4442         \pgftransformshift
4443         {
4444             \pgfpointlineattime { 0.5 }
4445             { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4446             { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4447         }
4448         \fp_set:Nn \l_tmpa_fp
4449         {
4450             atand
4451             (
4452                 \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4453                 \l_@@_x_final_dim - \l_@@_x_initial_dim
4454             )
4455         }
4456         \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4457         \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4458         \pgfnode
4459         { rectangle }
4460         { south }
4461         {
4462             \rotatebox { \fp_eval:n { - \l_tmpa_fp } } {
4463                 {
4464                     \c_math_toggle_token
4465                     \scriptstyle \l_@@_xdots_up_tl
4466                     \c_math_toggle_token
4467                 }
4468             }
4469         { }
4470         { \pgfusepath { } }
4471         \pgfnode
4472         { rectangle }
4473         { north }
4474         {

```

```

4475         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4476         {
4477             \c_math_toggle_token
4478             \scriptstyle \l_@@_xdots_down_tl
4479             \c_math_toggle_token
4480         }
4481     }
4482     { }
4483     { \pgfusepath { } }
4484 \endpgfscope
4485 }
4486 \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.

```

4487 \dim_zero_new:N \l_@@_l_dim
4488 \dim_set:Nn \l_@@_l_dim
4489 {
4490     \fp_to_dim:n
4491     {
4492         sqrt
4493         (
4494             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4495             +
4496             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4497         )
4498     }
4499 }

```

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.

```

4500 \bool_lazy_or:nnF
4501 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4502 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4503 \@@_draw_standard_dotted_line_i:
4504 \group_end:
4505 }
4506 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4507 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4508 {

```

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

```

4509 \bool_if:NTF \l_@@_initial_open_bool
4510 {
4511     \bool_if:NTF \l_@@_final_open_bool
4512     {
4513         \int_set:Nn \l_tmpa_int
4514         { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
4515     }
4516     {
4517         \int_set:Nn \l_tmpa_int
4518         {
4519             \dim_ratio:nn
4520             { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
4521             \l_@@_xdots_inter_dim
4522         }
4523     }
4524 }
4525 {
4526     \bool_if:NTF \l_@@_final_open_bool
4527     {

```

```

4528         \int_set:Nn \l_tmpa_int
4529         {
4530             \dim_ratio:nn
4531             { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
4532             \l_@@_xdots_inter_dim
4533         }
4534     }
4535     {
4536         \int_set:Nn \l_tmpa_int
4537         {
4538             \dim_ratio:nn
4539             {
4540                 \l_@@_l_dim
4541                 - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4542             }
4543             \l_@@_xdots_inter_dim
4544         }
4545     }
4546 }

```

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

```

4547 \dim_set:Nn \l_tmpa_dim
4548 {
4549     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4550     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4551 }
4552 \dim_set:Nn \l_tmpb_dim
4553 {
4554     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4555     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4556 }

```

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.

```

4557 \dim_gadd:Nn \l_@@_x_initial_dim
4558 {
4559     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4560     \dim_ratio:nn
4561     {
4562         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4563         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4564     }
4565     { 2 \l_@@_l_dim }
4566 }
4567 \dim_gadd:Nn \l_@@_y_initial_dim
4568 {
4569     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4570     \dim_ratio:nn
4571     {
4572         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4573         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4574     }
4575     { 2 \l_@@_l_dim }
4576 }
4577 \pgf@relevantforpicturesizefalse
4578 \int_step_inline:nnn 0 \l_tmpa_int
4579 {
4580     \pgfpathcircle
4581     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4582     { \l_@@_xdots_radius_dim }
4583     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4584     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4585 }

```

```

4586 \pgfusepathqfill
4587 }

```

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.

```

4588 \hook_gput_code:nnn { begindocument } { . }
4589 {
4590   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
4591   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4592   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4593   {
4594     \int_compare:nNnTF \c@jCol = 0
4595     { \@@_error:nn { in~first~col } \Ldots }
4596     {
4597       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4598       { \@@_error:nn { in~last~col } \Ldots }
4599       {
4600         \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
4601         { #1 , down = #2 , up = #3 }
4602       }
4603     }
4604     \bool_if:NF \l_@@_nullify_dots_bool
4605     { \phantom { \ensuremath { \@@_old_ldots } } }
4606     \bool_gset_true:N \g_@@_empty_cell_bool
4607   }

```

```

4608   \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4609   {
4610     \int_compare:nNnTF \c@jCol = 0
4611     { \@@_error:nn { in~first~col } \Cdots }
4612     {
4613       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4614       { \@@_error:nn { in~last~col } \Cdots }
4615       {
4616         \@@_instruction_of_type:nnn \c_false_bool { \Cdots }
4617         { #1 , down = #2 , up = #3 }
4618       }
4619     }
4620     \bool_if:NF \l_@@_nullify_dots_bool
4621     { \phantom { \ensuremath { \@@_old_cdots } } }
4622     \bool_gset_true:N \g_@@_empty_cell_bool
4623   }

```

```

4624   \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4625   {
4626     \int_compare:nNnTF \c@iRow = 0
4627     { \@@_error:nn { in~first~row } \Vdots }
4628     {

```

```

4629         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4630         { \@@_error:nn { in~last~row } \Vdots }
4631         {
4632             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4633             { #1 , down = #2 , up = #3 }
4634         }
4635     }
4636     \bool_if:NF \l_@@_nullify_dots_bool
4637     { \phantom { \ensuremath { \@@_old_vdots } } }
4638     \bool_gset_true:N \g_@@_empty_cell_bool
4639 }

4640 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4641 {
4642     \int_case:nnF \c@iRow
4643     {
4644         0 { \@@_error:nn { in~first~row } \Ddots }
4645         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4646     }
4647     {
4648         \int_case:nnF \c@jCol
4649         {
4650             0 { \@@_error:nn { in~first~col } \Ddots }
4651             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4652         }
4653         {
4654             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4655             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4656             { #1 , down = #2 , up = #3 }
4657         }
4658     }
4659 }
4660 \bool_if:NF \l_@@_nullify_dots_bool
4661 { \phantom { \ensuremath { \@@_old_ddots } } }
4662 \bool_gset_true:N \g_@@_empty_cell_bool
4663 }

4664 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4665 {
4666     \int_case:nnF \c@iRow
4667     {
4668         0 { \@@_error:nn { in~first~row } \Iddots }
4669         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4670     }
4671     {
4672         \int_case:nnF \c@jCol
4673         {
4674             0 { \@@_error:nn { in~first~col } \Iddots }
4675             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4676         }
4677         {
4678             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4679             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4680             { #1 , down = #2 , up = #3 }
4681         }
4682     }
4683     \bool_if:NF \l_@@_nullify_dots_bool
4684     { \phantom { \ensuremath { \@@_old_iddots } } }
4685     \bool_gset_true:N \g_@@_empty_cell_bool
4686 }
4687 }

```


End of the `\AddToHook`.

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

```

4688 \keys_define:nn { NiceMatrix / Ddots }
4689 {
4690   draw-first .bool_set:N = \l_@@_draw_first_bool ,
4691   draw-first .default:n = true ,
4692   draw-first .value_forbidden:n = true
4693 }

```

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

```

4694 \cs_new_protected:Npn \@@_Hspace:
4695 {
4696   \bool_gset_true:N \g_@@_empty_cell_bool
4697   \hspace
4698 }

```

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`.

```

4699 \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`.

```

4700 \cs_new:Npn \@@_Hdotsfor:
4701 {
4702   \bool_lazy_and:nnTF
4703     { \int_compare_p:nNn \c@jCol = 0 }
4704     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4705     {
4706       \bool_if:NTF \g_@@_after_col_zero_bool
4707       {
4708         \multicolumn { 1 } { c } { }
4709         \@@_Hdotsfor_i
4710       }
4711       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4712     }
4713     {
4714       \multicolumn { 1 } { c } { }
4715       \@@_Hdotsfor_i
4716     }
4717 }

```

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:`).

```

4718 \hook_gput_code:nnn { begindocument } { . }
4719 {
4720   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4721   \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.

```

4722   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4723   {
4724     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4725     {
4726       \@@_Hdotsfor:nnnn
4727       { \int_use:N \c@iRow }
4728       { \int_use:N \c@jCol }
4729       { #2 }

```

```

4730         {
4731             #1 , #3 ,
4732             down = \exp_not:n { #4 } ,
4733             up = \exp_not:n { #5 }
4734         }
4735     }
4736     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4737 }
4738 }

```

```

4739 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4740 {
4741     \bool_set_false:N \l_@@_initial_open_bool
4742     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4743     \int_set:Nn \l_@@_initial_i_int { #1 }
4744     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4745     \int_compare:nNnTF { #2 } = 1
4746     {
4747         \int_set:Nn \l_@@_initial_j_int 1
4748         \bool_set_true:N \l_@@_initial_open_bool
4749     }
4750     {
4751         \cs_if_exist:cTF
4752         {
4753             pgf @ sh @ ns @ \@@_env:
4754             - \int_use:N \l_@@_initial_i_int
4755             - \int_eval:n { #2 - 1 }
4756         }
4757         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4758         {
4759             \int_set:Nn \l_@@_initial_j_int { #2 }
4760             \bool_set_true:N \l_@@_initial_open_bool
4761         }
4762     }
4763     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4764     {
4765         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4766         \bool_set_true:N \l_@@_final_open_bool
4767     }
4768     {
4769         \cs_if_exist:cTF
4770         {
4771             pgf @ sh @ ns @ \@@_env:
4772             - \int_use:N \l_@@_final_i_int
4773             - \int_eval:n { #2 + #3 }
4774         }
4775         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4776         {
4777             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4778             \bool_set_true:N \l_@@_final_open_bool
4779         }
4780     }
4781     \group_begin:
4782     \int_compare:nNnTF { #1 } = 0
4783     { \color { nicematrix-first-row } }
4784     {
4785         \int_compare:nNnT { #1 } = \g_@@_row_total_int
4786         { \color { nicematrix-last-row } }
4787     }
4788     \keys_set:nn { NiceMatrix / xdots } { #4 }

```

```

4789 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4790 \@@_actually_draw_ldots:
4791 \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).

```

4792 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4793 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4794 }

4795 \hook_gput_code:nnn { begindocument } { . }
4796 {
4797   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4798   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4799   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4800   {
4801     \bool_gset_true:N \g_@@_empty_cell_bool % added
4802     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4803     {
4804       \@@_Vdotsfor:nnnn
4805       { \int_use:N \c@iRow }
4806       { \int_use:N \c@jCol }
4807       { #2 }
4808       {
4809         #1 , #3 ,
4810         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4811       }
4812     }
4813   }
4814 }

```

Enf of `\AddToHook`.

```

4815 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4816 {
4817   \bool_set_false:N \l_@@_initial_open_bool
4818   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

4819 \int_set:Nn \l_@@_initial_j_int { #2 }
4820 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4821 \int_compare:nNnTF #1 = 1
4822 {
4823   \int_set:Nn \l_@@_initial_i_int 1
4824   \bool_set_true:N \l_@@_initial_open_bool
4825 }
4826 {
4827   \cs_if_exist:cTF
4828   {
4829     pgf @ sh @ ns @ \@@_env:
4830     - \int_eval:n { #1 - 1 }
4831     - \int_use:N \l_@@_initial_j_int
4832   }
4833   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4834   {
4835     \int_set:Nn \l_@@_initial_i_int { #1 }
4836     \bool_set_true:N \l_@@_initial_open_bool
4837   }
4838 }
4839 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4840 {

```

```

4841 \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4842 \bool_set_true:N \l_@@_final_open_bool
4843 }
4844 {
4845 \cs_if_exist:cTF
4846 {
4847 pgf @ sh @ ns @ \@@_env:
4848 - \int_eval:n { #1 + #3 }
4849 - \int_use:N \l_@@_final_j_int
4850 }
4851 { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4852 {
4853 \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4854 \bool_set_true:N \l_@@_final_open_bool
4855 }
4856 }
4857 \group_begin:
4858 \int_compare:nNnTF { #2 } = 0
4859 { \color { nicematrix-first-col } }
4860 {
4861 \int_compare:nNnT { #2 } = \g_@@_col_total_int
4862 { \color { nicematrix-last-col } }
4863 }
4864 \keys_set:nn { NiceMatrix / xdots } { #4 }
4865 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4866 \@@_actually_draw_Vdots:
4867 \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).

```

4868 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4869 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4870 }

```

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

```

4871 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

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

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

First, we write a command with the following behaviour:

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

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

```

4872 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop

```

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

```

4873 {
4874   \tl_if_empty:nTF { #2 }
4875     { #1 }
4876     { \@@_double_int_eval:n #1-#2 \q_stop }
4877 }
4878 \cs_new:Npn \@@_double_int_eval:n #1-#2- \q_stop
4879 { \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).

```

4880 \hook_gput_code:nnn { begindocument } { . }
4881 {
4882   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4883   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4884   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4885     {
4886       \group_begin:
4887       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4888       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4889       \use:e
4890       {
4891         \@@_line_i:nn
4892         { \@@_double_int_eval:n #2 - \q_stop }
4893         { \@@_double_int_eval:n #3 - \q_stop }
4894       }
4895       \group_end:
4896     }
4897 }
4898 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4899 {
4900   \bool_set_false:N \l_@@_initial_open_bool
4901   \bool_set_false:N \l_@@_final_open_bool
4902   \bool_if:nTF
4903     {
4904       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4905       ||
4906       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4907     }
4908     {
4909       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4910     }

```

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

```

4911   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
4912 }
4913 \hook_gput_code:nnn { begindocument } { . }
4914 {
4915   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4916   {

```

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:`.

```

4917     \c_@@_pgfortikzpicture_tl
4918     \@@_draw_line_iii:nn { #1 } { #2 }
4919     \c_@@_endpgfortikzpicture_tl
4920   }
4921 }

```

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

```

4922 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2

```

```

4923 {
4924   \pgfrememberpicturepositiononpagetrue
4925   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4926   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4927   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4928   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4929   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4930   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4931   \@@_draw_line:
4932 }

```

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`

```

4933 \keys_define:nn { NiceMatrix / RowStyle }
4934 {
4935   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4936   cell-space-top-limit .initial:n = \c_zero_dim ,
4937   cell-space-top-limit .value_required:n = true ,
4938   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4939   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4940   cell-space-bottom-limit .value_required:n = true ,
4941   cell-space-limits .meta:n =
4942   {
4943     cell-space-top-limit = #1 ,
4944     cell-space-bottom-limit = #1 ,
4945   } ,
4946   color .tl_set:N = \l_@@_color_tl ,
4947   color .value_required:n = true ,
4948   bold .bool_set:N = \l_tmpa_bool ,
4949   bold .default:n = true ,
4950   bold .initial:n = false ,
4951   nb-rows .code:n =
4952   \str_if_eq:nnTF { #1 } { * }
4953   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
4954   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
4955   nb-rows .value_required:n = true ,
4956   rowcolor .tl_set:N = \l_tmpa_tl ,
4957   rowcolor .value_required:n = true ,
4958   rowcolor .initial:n = ,
4959   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
4960 }

4961 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4962 {
4963   \group_begin:
4964   \tl_clear:N \l_tmpa_tl % value of \rowcolor
4965   \tl_clear:N \l_@@_color_tl
4966   \int_set:Nn \l_@@_key_nb_rows_int 1
4967   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

If the key rowcolor has been used.
4968   \tl_if_empty:NF \l_tmpa_tl
4969   {

First, the end of the current row (we remind that \RowStyle applies to the end of the current row).
4970   \tl_gput_right:Nx \g_@@_pre_code_before_tl
4971   {

```

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

```

4972     \@@_exp_color_arg:NV \@@_rectanglecolor \l_tmpa_tl
4973     { \int_use:N \c@iRow - \int_use:N \c@jCol }
4974     { \int_use:N \c@iRow - * }
4975 }

```

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

```

4976     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4977     {
4978         \tl_gput_right:Nx \g_@@_pre_code_before_tl
4979         {
4980             \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
4981             {
4982                 \int_eval:n { \c@iRow + 1 }
4983                 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4984             }
4985         }
4986     }
4987 }
4988 \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4989 \tl_gput_right:Nx \g_@@_row_style_tl
4990 { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4991 \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

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

```

4992     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4993     {
4994         \tl_gput_right:Nx \g_@@_row_style_tl
4995         {
4996             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4997             {
4998                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
4999                 { \dim_use:N \l_tmpa_dim }
5000             }
5001         }
5002     }

```

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

```

5003     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5004     {
5005         \tl_gput_right:Nx \g_@@_row_style_tl
5006         {
5007             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5008             {
5009                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5010                 { \dim_use:N \l_tmpb_dim }
5011             }
5012         }
5013     }

```

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

```

5014     \tl_if_empty:NF \l_@@_color_tl
5015     {
5016         \tl_gput_right:Nx \g_@@_row_style_tl
5017         {
5018             \mode_leave_vertical:
5019             \@@_color:n { \l_@@_color_tl }
5020         }
5021     }

```

`\l_tmpa_bool` is the value of the key `bold`.

```

5022     \bool_if:NT \l_tmpa_bool
5023     {
5024         \tl_gput_right:Nn \g_@@_row_style_tl
5025         {
5026             \if_mode_math:
5027                 \c_math_toggle_token

```

```

5028         \bfseries \boldmath
5029         \c_math_toggle_token
5030     \else:
5031         \bfseries \boldmath
5032     \fi:
5033 }
5034 }
5035 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5036 \group_end:
5037 \g_@@_row_style_tl
5038 \ignorespaces
5039 }

```

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).

```

5040 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5041 {

```

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.

```

5042     \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`.

```

5043     \str_if_in:nnF { #1 } { !! }
5044     {
5045         \seq_map_indexed_inline:Nn \g_@@_colors_seq
5046         { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5047     }
5048     \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

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

```

5049     {
5050         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5051         \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5052     }

```

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

```

5053     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5054 }

```



```

5055 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5056 \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`).

```

5057 \cs_new_protected:Npn \@@_actually_color:
5058 {
5059   \pgfpicture
5060   \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.

```

5061   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5062   {
5063     \pgfsetcornersarced
5064     {
5065       \pgfpoint
5066       { \l_@@_tab_rounded_corners_dim }
5067       { \l_@@_tab_rounded_corners_dim }
5068     }
5069   %   \end{macrocode}
5070   % Because we want \pkg{nicematrix} compatible with arrays constructed by
5071   % \pkg{array}, the nodes for the rows and columns (that is to say the nodes
5072   % |row-|\textsl{i} and |col-|\textsl{j}|) have not always the expected position,
5073   % that is to say, there is sometimes a slight shifting of something such as
5074   % |\arrayrulewidth|. Now, for the clipping, we have to change slightly the
5075   % position of that clipping whether a rounded rectangle around the array is
5076   % required. That's the point which is tested in the following line.
5077   %   \begin{macrocode}
5078   \bool_if:NTF \l_@@_hvlines_bool
5079   {
5080     \pgfpathrectanglecorners
5081     {
5082       \pgfpointadd
5083       { \@@_qpoint:n { row-1 } }
5084       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5085     }
5086     {
5087       \pgfpointadd
5088       {
5089         \@@_qpoint:n
5090         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5091       }
5092       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5093     }
5094   }
5095   {
5096     \pgfpathrectanglecorners
5097     { \@@_qpoint:n { row-1 } }
5098     {
5099       \pgfpointadd
5100       {
5101         \@@_qpoint:n
5102         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5103       }
5104       { \pgfpoint \c_zero_dim \arrayrulewidth }
5105     }
5106   }
5107   \pgfusepath { clip }
5108 }
5109 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5110 {
5111   \begin { pgfscope }

```

```

5112         \@@_color_opacity ##2
5113         \use:c { g_@@_color _ ##1 _t1 }
5114         \tl_gclear:c { g_@@_color _ ##1 _t1 }
5115         \pgfusepath { fill }
5116     \end { pgfscope }
5117 }
5118 \endpgfpicture
5119 }

```

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

```

5120 \cs_new_protected:Npn \@@_color_opacity
5121 {
5122     \peek_meaning:NTF [
5123         { \@@_color_opacity:w }
5124         { \@@_color_opacity:w [ ] }
5125     }

```

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.

```

5126 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5127 {
5128     \tl_clear:N \l_tmpa_tl
5129     \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

`\l_tmpa_tl` (if not empty) is now the opacity and `\l_tmpb_tl` (if not empty) is now the colorimetric space.

```

5130     \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5131     \tl_if_empty:NTF \l_tmpb_tl
5132         { \@declaredcolor }
5133         { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5134 }

```

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

```

5135 \keys_define:nn { nicematrix / color-opacity }
5136 {
5137     opacity .tl_set:N          = \l_tmpa_tl ,
5138     opacity .value_required:n = true
5139 }

5140 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5141 {
5142     \tl_set:Nn \l_@@_rows_tl { #1 }
5143     \tl_set:Nn \l_@@_cols_tl { #2 }
5144     \@@_cartesian_path:
5145 }

```

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

```

5146 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5147 {
5148     \tl_if_blank:NF { #2 }
5149     {
5150         \@@_add_to_colors_seq:xn
5151         { \tl_if_blank:NF { #1 } { [ #1 ] } { #2 } }
5152         { \@@_cartesian_color:nn { #3 } { - } }
5153     }
5154 }

```

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

```

5155 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5156 {
5157   \tl_if_blank:nF { #2 }
5158   {
5159     \@@_add_to_colors_seq:xn
5160     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5161     { \@@_cartesian_color:nn { - } { #3 } }
5162   }
5163 }

```

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

```

5164 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5165 {
5166   \tl_if_blank:nF { #2 }
5167   {
5168     \@@_add_to_colors_seq:xn
5169     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5170     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5171   }
5172 }

```

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

```

5173 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5174 {
5175   \tl_if_blank:nF { #2 }
5176   {
5177     \@@_add_to_colors_seq:xn
5178     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5179     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5180   }
5181 }

```

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

```

5182 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5183 {
5184   \@@_cut_on_hyphen:w #1 \q_stop
5185   \tl_clear_new:N \l_@@_tmpc_tl
5186   \tl_clear_new:N \l_@@_tmpd_tl
5187   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5188   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5189   \@@_cut_on_hyphen:w #2 \q_stop
5190   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5191   \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.

```

5192   \@@_cartesian_path:n { #3 }
5193 }

```

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

```

5194 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5195 {
5196   \clist_map_inline:nn { #3 }
5197   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5198 }

```

```

5199 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5200 {
5201   \int_step_inline:nn { \int_use:N \c@iRow }

```

```

5202 {
5203   \int_step_inline:nn { \int_use:N \c@jCol }
5204   {
5205     \int_if_even:nTF { ####1 + #1 }
5206     { \@@_cellcolor [ #1 ] { #2 } }
5207     { \@@_cellcolor [ #1 ] { #3 } }
5208     { ##1 - ####1 }
5209   }
5210 }
5211 }

```

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”.

```

5212 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5213 {
5214   \@@_rectanglecolor [ #1 ] { #2 }
5215   { 1 - 1 }
5216   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5217 }

5218 \keys_define:nn { NiceMatrix / rowcolors }
5219 {
5220   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5221   respect-blocks .default:n = true ,
5222   cols .tl_set:N = \l_@@_cols_tl ,
5223   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5224   restart .default:n = true ,
5225   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5226 }

```

The command `\rowcolors` (accessible in the `code-before`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`. Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

#1 (optional) is the color space ; **#2** is a list of intervals of rows ; **#3** is the list of colors ; **#4** is for the optional list of pairs *key=value*.

```

5227 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5228 {

```

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

```

5229   \group_begin:
5230   \seq_clear_new:N \l_@@_colors_seq
5231   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5232   \tl_clear_new:N \l_@@_cols_tl
5233   \tl_set:Nn \l_@@_cols_tl { - }
5234   \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).

```

5235   \int_zero_new:N \l_@@_color_int
5236   \int_set:Nn \l_@@_color_int 1
5237   \bool_if:NT \l_@@_respect_blocks_bool
5238   {

```

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`).

```

5239   \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5240   \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5241   { \@@_not_in_exterior_p:nnnnn #1 }
5242 }

```

```

5243 \pgfpicture
5244 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5245 \clist_map_inline:nn { #2 }
5246 {
5247   \tl_set:Nn \l_tmpa_tl { ##1 }
5248   \tl_if_in:NnTF \l_tmpa_tl { - }
5249   { \@@_cut_on_hyphen:w ##1 \q_stop }
5250   { \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.

```

5251   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5252   \bool_if:NNTF \l_@@_rowcolors_restart_bool
5253   { \int_set:Nn \l_@@_color_int 1 }
5254   { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
5255   \int_zero_new:N \l_@@_tmpc_int
5256   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5257   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5258   {

```

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

```

5259       \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).

```

5260       \bool_if:NT \l_@@_respect_blocks_bool
5261       {
5262         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5263         { \@@_intersect_our_row_p:nnnnn ###1 }
5264         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

5265       }
5266       \tl_set:Nx \l_@@_rows_tl
5267       { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5268       \tl_clear_new:N \l_@@_color_tl
5269       \tl_set:Nx \l_@@_color_tl
5270       {
5271         \@@_color_index:n
5272         {
5273           \int_mod:nn
5274           { \l_@@_color_int - 1 }
5275           { \seq_count:N \l_@@_colors_seq }
5276           + 1
5277         }
5278       }
5279       \tl_if_empty:NF \l_@@_color_tl
5280       {
5281         \@@_add_to_colors_seq:xx
5282         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5283         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5284       }
5285       \int_incr:N \l_@@_color_int
5286       \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5287     }
5288   }
5289 \endpgfpicture
5290 \group_end:
5291 }

```

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.

```

5292 \cs_new:Npn \@@_color_index:n #1

```

```

5293 {
5294   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5295   { \@@_color_index:n { #1 - 1 } }
5296   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5297 }

```

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

```

5298 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
5299 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

5300 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5301 {
5302   \int_compare:nNnT { #3 } > \l_tmpb_int
5303   { \int_set:Nn \l_tmpb_int { #3 } }
5304 }

5305 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5306 {
5307   \bool_lazy_or:nnTF
5308   { \int_compare_p:nNn { #4 } = \c_zero_int }
5309   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5310   \prg_return_false:
5311   \prg_return_true:
5312 }

```

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

```

5313 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5314 {
5315   \bool_if:nTF
5316   {
5317     \int_compare_p:n { #1 <= \l_tmpa_int }
5318     &&
5319     \int_compare_p:n { \l_tmpa_int <= #3 }
5320   }
5321   \prg_return_true:
5322   \prg_return_false:
5323 }

```

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`).

```

5324 \cs_new_protected:Npn \@@_cartesian_path:n #1
5325 {
5326   \bool_lazy_and:nnT
5327   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5328   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5329   {
5330     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5331     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5332   }

```

We begin the loop over the columns.

```

5333 \clist_map_inline:Nn \l_@@_cols_tl
5334 {
5335   \tl_set:Nn \l_tmpa_tl { ##1 }
5336   \tl_if_in:NnTF \l_tmpa_tl { - }

```

```

5337 { \@@_cut_on_hyphen:w ##1 \q_stop }
5338 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5339 \bool_lazy_or:nnT
5340 { \tl_if_blank_p:V \l_tmpa_tl }
5341 { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5342 { \tl_set:Nn \l_tmpa_tl { 1 } }
5343 \bool_lazy_or:nnT
5344 { \tl_if_blank_p:V \l_tmpb_tl }
5345 { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5346 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5347 \int_compare:nNnT \l_tmpb_tl > \c@jCol
5348 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

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

```

5349 \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.

```

5350 \@@_qpoint:n { col - \l_tmpa_tl }
5351 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5352 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5353 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5354 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5355 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5356 \clist_map_inline:Nn \l_@@_rows_tl
5357 {
5358   \tl_set:Nn \l_tmpa_tl { #####1 }
5359   \tl_if_in:NnTF \l_tmpa_tl { - }
5360   { \@@_cut_on_hyphen:w #####1 \q_stop }
5361   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5362   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5363   \tl_if_empty:NT \l_tmpb_tl
5364   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5365   \int_compare:nNnT \l_tmpb_tl > \c@iRow
5366   { \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`.

```

5367 \seq_if_in:NxF \l_@@_corners_cells_seq
5368 { \l_tmpa_tl - \l_@@_tmpc_tl }
5369 {
5370   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5371   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5372   \@@_qpoint:n { row - \l_tmpa_tl }
5373   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5374   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5375   \pgfpathrectanglecorners
5376   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5377   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5378 }
5379 }
5380 }
5381 }

```

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`).

```

5382 \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 `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

5383 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5384 {
5385   \clist_set_eq:NN \l_tmpa_clist #1
5386   \clist_clear:N #1
5387   \clist_map_inline:Nn \l_tmpa_clist
5388   {
5389     \tl_set:Nn \l_tmpa_tl { ##1 }
5390     \tl_if_in:NnTF \l_tmpa_tl { - }
5391     { \@@_cut_on_hyphen:w ##1 \q_stop }
5392     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5393     \bool_lazy_or:nnT
5394     { \tl_if_blank_p:V \l_tmpa_tl }
5395     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5396     { \tl_set:Nn \l_tmpa_tl { 1 } }
5397     \bool_lazy_or:nnT
5398     { \tl_if_blank_p:V \l_tmpb_tl }
5399     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5400     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5401     \int_compare:nNnT \l_tmpb_tl > #2
5402     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5403     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5404     { \clist_put_right:Nn #1 { #####1 } }
5405   }
5406 }

```

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

```

5407 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5408 {
5409   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5410   {

```

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`).

```

5411     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5412     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5413   }
5414   \ignorespaces
5415 }

```

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

```

5416 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5417 {
5418   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5419   {
5420     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5421     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5422     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5423   }
5424   \ignorespaces
5425 }

```

```

5426 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5427 {

```

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).

```

5428   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5429   {

```


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).

```

5430     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5431     {
5432         \exp_not:N \columncolor [ #1 ]
5433         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5434     }
5435 }
5436 }
```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumntype` 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`.

```

5437 \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`.

```

5438 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5439 {
5440     \int_compare:nNnTF \l_@@_first_col_int = 0
5441     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5442     {
5443         \int_compare:nNnTF \c@jCol = 0
5444         {
5445             \int_compare:nNnF \c@iRow = { -1 }
5446             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5447         }
5448         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5449     }
5450 }
```

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.

```

5451 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5452 {
5453     \int_compare:nNnF \c@iRow = 0
5454     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
5455 }
```

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.

```

5456 \keys_define:nn { NiceMatrix / Rules }
5457 {
5458   position .int_set:N = \l_@@_position_int ,
5459   position .value_required:n = true ,
5460   start .int_set:N = \l_@@_start_int ,
5461   start .initial:n = 1 ,
5462   end .code:n =
5463     \bool_lazy_or:nnTF
5464       { \tl_if_empty_p:n { #1 } }
5465       { \str_if_eq_p:nn { #1 } { last } }
5466       { \int_set_eq:NN \l_@@_end_int \c@jCol }
5467       { \int_set:Nn \l_@@_end_int { #1 } }
5468 }

```

It’s possible that the rule won’t be drawn continuously from `start` ot `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

5469 \keys_define:nn { NiceMatrix / RulesBis }
5470 {
5471   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5472   multiplicity .initial:n = 1 ,
5473   dotted .bool_set:N = \l_@@_dotted_bool ,
5474   dotted .initial:n = false ,
5475   dotted .default:n = true ,
5476   color .code:n = \@@_set_CT@arc@:n { #1 } ,
5477   color .value_required:n = true ,
5478   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5479   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.

```

5480   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5481   tikz .value_required:n = true ,
5482   tikz .initial:n = ,
5483   total-width .dim_set:N = \l_@@_rule_width_dim ,
5484   total-width .value_required:n = true ,
5485   width .meta:n = { total-width = #1 } ,
5486   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5487 }

```

The vertical rules

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

```

5488 \cs_new_protected:Npn \@@_vline:n #1
5489 {

```

The group is for the options.

```

5490   \group_begin:
5491   \int_zero_new:N \l_@@_end_int
5492   \int_set_eq:NN \l_@@_end_int \c@iRow
5493   \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).

```

5494 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5495 \@@_vline_i:
5496 \group_end:
5497 }

5498 \cs_new_protected:Npn \@@_vline_i:
5499 {
5500 \int_zero_new:N \l_@@_local_start_int
5501 \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`.

```

5502 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5503 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5504 \l_tmpa_tl
5505 {

```

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.

```

5506 \bool_gset_true:N \g_tmpa_bool
5507 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5508 { \@@_test_vline_in_block:nnnnn ##1 }
5509 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5510 { \@@_test_vline_in_block:nnnnn ##1 }
5511 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5512 { \@@_test_vline_in_stroken_block:nnnn ##1 }
5513 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5514 \bool_if:NTF \g_tmpa_bool
5515 {
5516 \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.

```

5517 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5518 }
5519 {
5520 \int_compare:nNnT \l_@@_local_start_int > 0
5521 {
5522 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5523 \@@_vline_ii:
5524 \int_zero:N \l_@@_local_start_int
5525 }
5526 }
5527 }
5528 \int_compare:nNnT \l_@@_local_start_int > 0
5529 {
5530 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5531 \@@_vline_ii:
5532 }
5533 }

```

```

5534 \cs_new_protected:Npn \@@_test_in_corner_v:
5535 {
5536 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5537 {
5538 \seq_if_in:NxT
5539 \l_@@_corners_cells_seq
5540 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5541 { \bool_set_false:N \g_tmpa_bool }
5542 }

```

```

5543 {
5544   \seq_if_in:NxT
5545   \l_@@_corners_cells_seq
5546   { \l_tmpa_tl - \l_tmpb_tl }
5547   {
5548     \int_compare:nNnTF \l_tmpb_tl = 1
5549     { \bool_set_false:N \g_tmpa_bool }
5550     {
5551       \seq_if_in:NxT
5552       \l_@@_corners_cells_seq
5553       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5554       { \bool_set_false:N \g_tmpa_bool }
5555     }
5556   }
5557 }
5558 }

5559 \cs_new_protected:Npn \@@_vline_ii:
5560 {
5561   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5562   \bool_if:NTF \l_@@_dotted_bool
5563   \@@_vline_iv:
5564   {
5565     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5566     \@@_vline_iii:
5567     \@@_vline_v:
5568   }
5569 }

```

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

```

5570 \cs_new_protected:Npn \@@_vline_iii:
5571 {
5572   \pgfpicture
5573   \pgfrememberpicturerepositiononpagetrue
5574   \pgf@relevantforpicturesizefalse
5575   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5576   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5577   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5578   \dim_set:Nn \l_tmpb_dim
5579   {
5580     \pgf@x
5581     - 0.5 \l_@@_rule_width_dim
5582     +
5583     ( \arrayrulewidth * \l_@@_multiplicity_int
5584       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5585   }
5586   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5587   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5588   \bool_lazy_all:nT
5589   {
5590     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5591     { \cs_if_exist_p:N \CT@drsc@ }
5592     { ! \tl_if_blank_p:V \CT@drsc@ }
5593   }
5594   {
5595     \group_begin:
5596     \CT@drsc@
5597     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5598     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5599     \dim_set:Nn \l_@@_tmpd_dim
5600     {
5601       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )

```

```

5602         * ( \l_@@_multiplicity_int - 1 )
5603     }
5604     \pgfpathrectanglecorners
5605     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5606     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5607     \pgfusepath { fill }
5608     \group_end:
5609 }
5610 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5611 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5612 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5613 {
5614     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5615     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5616     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5617     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5618 }
5619 \CT@arc@
5620 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5621 \pgfsetrectcap
5622 \pgfusepathqstroke
5623 \endpgfpicture
5624 }

```

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

```

5625 \cs_new_protected:Npn \@@_vline_iv:
5626 {
5627     \pgfpicture
5628     \pgfrememberpicturepositiononpagetrue
5629     \pgf@relevantforpicturesizefalse
5630     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5631     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5632     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5633     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5634     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5635     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5636     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5637     \CT@arc@
5638     \@@_draw_line:
5639     \endpgfpicture
5640 }

```

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`).

```

5641 \cs_new_protected:Npn \@@_vline_v:
5642 {
5643     \begin {tikzpicture }
5644     \pgfrememberpicturepositiononpagetrue
5645     \pgf@relevantforpicturesizefalse
5646     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5647     \dim_set_eq:NN \l_tmpa_dim \pgf@y
5648     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5649     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5650     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5651     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5652     \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5653     \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5654     ( \l_tmpb_dim , \l_tmpa_dim ) --
5655     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5656     \end { tikzpicture }
5657 }

```

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).

```

5658 \cs_new_protected:Npn \@@_draw_vlines:
5659 {
5660   \int_step_inline:nnn
5661     {
5662       \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5663         1 2
5664     }
5665     {
5666       \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5667         { \int_eval:n { \c@jCol + 1 } }
5668         \c@jCol
5669     }
5670     {
5671       \tl_if_eq:NnF \l_@@_vlines_clist { all }
5672       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5673       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5674     }
5675 }

```

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}`.

```

5676 \cs_new_protected:Npn \@@_hline:n #1
5677 {

```

The group is for the options.

```

5678   \group_begin:
5679   \int_zero_new:N \l_@@_end_int
5680   \int_set_eq:NN \l_@@_end_int \c@jCol
5681   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5682   \@@_hline_i:
5683   \group_end:
5684 }
5685 \cs_new_protected:Npn \@@_hline_i:
5686 {
5687   \int_zero_new:N \l_@@_local_start_int
5688   \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`.

```

5689   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5690   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5691     \l_tmpb_tl
5692     {

```

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.

```

5693       \bool_gset_true:N \g_tmpa_bool
5694       \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5695         { \@@_test_hline_in_block:nnnnn ##1 }
5696       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5697         { \@@_test_hline_in_block:nnnnn ##1 }
5698       \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5699         { \@@_test_hline_in_stroken_block:nnnn ##1 }
5700       \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5701       \bool_if:NTF \g_tmpa_bool
5702         {
5703           \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.

```

5704         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5705     }
5706     {
5707         \int_compare:nNnT \l_@@_local_start_int > 0
5708         {
5709             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5710             \@@_hline_ii:
5711             \int_zero:N \l_@@_local_start_int
5712         }
5713     }
5714 }
5715 \int_compare:nNnT \l_@@_local_start_int > 0
5716 {
5717     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5718     \@@_hline_ii:
5719 }
5720 }

```

```

5721 \cs_new_protected:Npn \@@_test_in_corner_h:
5722 {
5723     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5724     {
5725         \seq_if_in:NxT
5726         \l_@@_corners_cells_seq
5727         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5728         { \bool_set_false:N \g_tmpa_bool }
5729     }
5730     {
5731         \seq_if_in:NxT
5732         \l_@@_corners_cells_seq
5733         { \l_tmpa_tl - \l_tmpb_tl }
5734         {
5735             \int_compare:nNnTF \l_tmpa_tl = 1
5736             { \bool_set_false:N \g_tmpa_bool }
5737             {
5738                 \seq_if_in:NxT
5739                 \l_@@_corners_cells_seq
5740                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5741                 { \bool_set_false:N \g_tmpa_bool }
5742             }
5743         }
5744     }
5745 }

```

```

5746 \cs_new_protected:Npn \@@_hline_ii:
5747 {
5748     % \bool_set_false:N \l_@@_dotted_bool
5749     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5750     \bool_if:NTF \l_@@_dotted_bool
5751     \@@_hline_iv:
5752     {
5753         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5754         \@@_hline_iii:
5755         \@@_hline_v:
5756     }
5757 }

```

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

```

5758 \cs_new_protected:Npn \@@_hline_iii:

```

```

5759 {
5760   \pgfpicture
5761   \pgfrememberpicturepositiononpagetrue
5762   \pgf@relevantforpicturesizefalse
5763   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5764   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5765   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5766   \dim_set:Nn \l_tmpb_dim
5767   {
5768     \pgf@y
5769     - 0.5 \l_@@_rule_width_dim
5770     +
5771     ( \arrayrulewidth * \l_@@_multiplicity_int
5772       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5773   }
5774   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5775   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5776   \bool_lazy_all:nT
5777   {
5778     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5779     { \cs_if_exist_p:N \CT@drsc@ }
5780     { ! \tl_if_blank_p:V \CT@drsc@ }
5781   }
5782   {
5783     \group_begin:
5784     \CT@drsc@
5785     \dim_set:Nn \l_@@_tmpd_dim
5786     {
5787       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5788       * ( \l_@@_multiplicity_int - 1 )
5789     }
5790     \pgfpathrectanglecorners
5791     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5792     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5793     \pgfusepathqfill
5794     \group_end:
5795   }
5796   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5797   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5798   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5799   {
5800     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5801     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5802     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5803     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5804   }
5805   \CT@arc@
5806   \pgfsetlinewidth { 1.1 \arrayrulewidth }
5807   \pgfsetrectcap
5808   \pgfusepathqstroke
5809   \endpgfpicture
5810 }

```

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}

```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.


```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

```

5811 \cs_new_protected:Npn \@@_hline_iv:
5812 {
5813   \pgfpicture
5814   \pgfrememberpicturepositiononpagetrue
5815   \pgf@relevantforpicturesizefalse
5816   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5817   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5818   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
5819   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5820   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5821   \int_compare:nNnT \l_@@_local_start_int = 1
5822   {
5823     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5824     \bool_if:NT \g_@@_NiceArray_bool
5825     { \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.

```

5826   \tl_if_eq:NnF \g_@@_left_delim_tl (
5827     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
5828   )
5829   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5830   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5831   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5832   {
5833     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5834     \bool_if:NT \g_@@_NiceArray_bool
5835     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5836     \tl_if_eq:NnF \g_@@_right_delim_tl )
5837     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5838   }
5839   \CT@arc@
5840   \@@_draw_line:
5841   \endpgfpicture
5842 }

```

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`).

```

5843 \cs_new_protected:Npn \@@_hline_v:
5844 {
5845   \begin { tikzpicture }
5846   \pgfrememberpicturepositiononpagetrue
5847   \pgf@relevantforpicturesizefalse
5848   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5849   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5850   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5851   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5852   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5853   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5854   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5855   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5856     ( \l_tmpa_dim , \l_tmpb_dim ) --
5857     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5858   \end { tikzpicture }
5859 }

```

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)).

```

5860 \cs_new_protected:Npn \@@_draw_hlines:
5861 {
5862   \int_step_inline:nnn
5863   {
5864     \bool_if:NTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5865     1 2
5866   }
5867   {
5868     \bool_if:NTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5869     { \int_eval:n { \c@iRow + 1 } }
5870     \c@iRow
5871   }
5872   {
5873     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5874     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5875     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5876   }
5877 }

```

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

```

5878 \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.

```

5879 \cs_set:Npn \@@_Hline_i:n #1
5880 {
5881   \peek_remove_spaces:n
5882   {
5883     \peek_meaning:NTF \Hline
5884     { \@@_Hline_ii:nn { #1 + 1 } }
5885     { \@@_Hline_iii:n { #1 } }
5886   }
5887 }
5888 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5889 \cs_set:Npn \@@_Hline_iii:n #1
5890 {
5891   \peek_meaning:NTF [
5892   { \@@_Hline_iv:nw { #1 } }
5893   { \@@_Hline_iv:nw { #1 } [ ] }
5894 }
5895 \cs_set:Npn \@@_Hline_iv:nw #1 [ #2 ]
5896 {
5897   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
5898   \skip_vertical:n { \l_@@_rule_width_dim }
5899   \tl_gput_right:Nx \g_@@_pre_code_after_tl
5900   {
5901     \@@_hline:n
5902     {
5903       multiplicity = #1 ,
5904       position = \int_eval:n { \c@iRow + 1 } ,
5905       total-width = \dim_use:N \l_@@_rule_width_dim ,
5906       #2
5907     }
5908   }
5909   \egroup
5910 }

```

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).

```
5911 \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`).

```
5912 \cs_new_protected:Npn \@@_custom_line:n #1
5913 {
5914   \str_clear_new:N \l_@@_command_str
5915   \str_clear_new:N \l_@@_ccommand_str
5916   \str_clear_new:N \l_@@_letter_str
5917   \keys_set_known:nnN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl
```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```
5918   \bool_lazy_all:nTF
5919     {
5920       { \str_if_empty_p:N \l_@@_letter_str }
5921       { \str_if_empty_p:N \l_@@_command_str }
5922       { \str_if_empty_p:N \l_@@_ccommand_str }
5923     }
5924     { \@@_error:n { No-letter~and~no-command } }
5925     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
5926   }

5927 \keys_define:nn { NiceMatrix / custom-line }
5928 {
5929   letter .str_set:N = \l_@@_letter_str ,
5930   letter .value_required:n = true ,
5931   command .str_set:N = \l_@@_command_str ,
5932   command .value_required:n = true ,
5933   ccommand .str_set:N = \l_@@_ccommand_str ,
5934   ccommand .value_required:n = true ,
5935 }
```

```
5936 \cs_new_protected:Npn \@@_custom_line_i:n #1
5937 {
```

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

```
5938   \bool_set_false:N \l_@@_tikz_rule_bool
5939   \bool_set_false:N \l_@@_dotted_rule_bool
5940   \bool_set_false:N \l_@@_color_bool

5941   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
5942   \bool_if:NT \l_@@_tikz_rule_bool
5943     {
5944       \IfPackageLoadedTF { tikz }
5945         { }
5946         { \@@_error:n { tikz~in~custom-line~without~tikz } }
5947       \bool_if:NT \l_@@_color_bool
5948         { \@@_error:n { color~in~custom-line~with~tikz } }
5949     }
```

```

5950 \bool_if:nT
5951 {
5952   \int_compare:nNn \l_@@_multiplicity_int > 1
5953   && \l_@@_dotted_rule_bool
5954 }
5955 { \@@_error:n { key~multiplicity~with~dotted } }
5956 \str_if_empty:NF \l_@@_letter_str
5957 {
5958   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
5959   { \@@_error:n { Several~letters } }
5960   {
5961     \exp_args:NnV \tl_if_in:NnTF
5962     \c_@@_forbidden_letters_str \l_@@_letter_str
5963     { \@@_error:n { Forbidden~letter } }
5964   }

```

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.

```

5965 \keys_define:nx { NiceMatrix / ColumnTypes }
5966 {
5967   \l_@@_letter_str .code:n =
5968   { \@@_v_custom_line:n { \exp_not:n { #1 } } }
5969 }
5970 }
5971 }
5972 }
5973 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
5974 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
5975 }
5976 \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.

```

5977 \keys_define:nn { NiceMatrix / custom-line-bis }
5978 {
5979   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5980   multiplicity .initial:n = 1 ,
5981   multiplicity .value_required:n = true ,
5982   color .code:n = \bool_set_true:N \l_@@_color_bool ,
5983   color .value_required:n = true ,
5984   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5985   tikz .value_required:n = true ,
5986   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5987   dotted .value_forbidden:n = true ,
5988   total-width .code:n = { } ,
5989   total-width .value_required:n = true ,
5990   width .code:n = { } ,
5991   width .value_required:n = true ,
5992   sep-color .code:n = { } ,
5993   sep-color .value_required:n = true ,
5994   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
5995 }

```

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

```

5996 \bool_new:N \l_@@_dotted_rule_bool
5997 \bool_new:N \l_@@_tikz_rule_bool
5998 \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`.

```

5999 \keys_define:nn { NiceMatrix / custom-line-width }
6000 {
6001     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6002     multiplicity .initial:n = 1 ,
6003     multiplicity .value_required:n = true ,
6004     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6005     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6006                     \bool_set_true:N \l_@@_total_width_bool ,
6007     total-width .value_required:n = true ,
6008     width .meta:n = { total-width = #1 } ,
6009     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6010 }

```

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).

```

6011 \cs_new_protected:Npn \@@_h_custom_line:n #1
6012 {

```

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.

```

6013     \cs_set:cpn { nicematrix - \l_@@_command_str }
6014     {
6015         \noalign
6016         {
6017             \@@_compute_rule_width:n { #1 }
6018             \skip_vertical:n { \l_@@_rule_width_dim }
6019             \tl_gput_right:Nx \g_@@_pre_code_after_tl
6020             {
6021                 \@@_hline:n
6022                 {
6023                     #1 ,
6024                     position = \int_eval:n { \c@iRow + 1 } ,
6025                     total-width = \dim_use:N \l_@@_rule_width_dim
6026                 }
6027             }
6028         }
6029     }
6030     \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6031 }
6032 \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).

```

6033 \cs_new_protected:Npn \@@_c_custom_line:n #1
6034 {

```

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

```

6035     \exp_args:Nc \NewExpandableDocumentCommand
6036     { nicematrix - \l_@@_ccommand_str }
6037     { 0 { } m }
6038     {
6039         \noalign
6040         {
6041             \@@_compute_rule_width:n { #1 , ##1 }
6042             \skip_vertical:n { \l_@@_rule_width_dim }
6043             \clist_map_inline:nn
6044             { ##2 }
6045             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6046         }

```

```

6047     }
6048     \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6049 }

```

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*.

```

6050 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6051 {
6052   \str_if_in:nnTF { #2 } { - }
6053   { \@@_cut_on_hyphen:w #2 \q_stop }
6054   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6055   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6056   {
6057     \@@_hline:n
6058     {
6059       #1 ,
6060       start = \l_tmpa_tl ,
6061       end = \l_tmpb_tl ,
6062       position = \int_eval:n { \c@iRow + 1 } ,
6063       total-width = \dim_use:N \l_@@_rule_width_dim
6064     }
6065   }
6066 }
6067 \cs_generate_variant:Nn \@@_c_custom_line:nn { n V }
6068 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6069 {
6070   \bool_set_false:N \l_@@_tikz_rule_bool
6071   \bool_set_false:N \l_@@_total_width_bool
6072   \bool_set_false:N \l_@@_dotted_rule_bool
6073   \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6074   \bool_if:NF \l_@@_total_width_bool
6075   {
6076     \bool_if:NTF \l_@@_dotted_rule_bool
6077     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6078     {
6079       \bool_if:NF \l_@@_tikz_rule_bool
6080       {
6081         \dim_set:Nn \l_@@_rule_width_dim
6082         {
6083           \arrayrulewidth * \l_@@_multiplicity_int
6084           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6085         }
6086       }
6087     }
6088   }
6089 }
6090 \cs_new_protected:Npn \@@_v_custom_line:n #1
6091 {
6092   \@@_compute_rule_width:n { #1 }

```

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

```

6093   \tl_gput_right:Nx \g_@@_preamble_tl
6094   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6095   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6096   {
6097     \@@_vline:n
6098     {
6099       #1 ,
6100       position = \int_eval:n { \c@jCol + 1 } ,
6101       total-width = \dim_use:N \l_@@_rule_width_dim
6102     }
6103   }
6104 }

```

```

6105 \@@_custom_line:n
6106 { 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.

```

6107 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6108 {
6109   \bool_lazy_all:nT
6110   {
6111     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6112     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6113     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6114     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6115   }
6116   { \bool_gset_false:N \g_tmpa_bool }
6117 }

```

The same for vertical rules.

```

6118 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6119 {
6120   \bool_lazy_all:nT
6121   {
6122     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6123     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6124     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6125     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6126   }
6127   { \bool_gset_false:N \g_tmpa_bool }
6128 }

6129 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6130 {
6131   \bool_lazy_all:nT
6132   {
6133     {
6134       ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6135       || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6136     }
6137     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6138     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6139   }
6140   { \bool_gset_false:N \g_tmpa_bool }
6141 }

6142 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6143 {
6144   \bool_lazy_all:nT
6145   {
6146     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6147     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6148     {
6149       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6150       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6151     }
6152   }
6153   { \bool_gset_false:N \g_tmpa_bool }
6154 }

```

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.

```
6155 \cs_new_protected:Npn \@@_compute_corners:
6156 {
```

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.

```
6157   \seq_clear_new:N \l_@@_corners_cells_seq
6158   \clist_map_inline:Nn \l_@@_corners_clist
6159   {
6160     \str_case:nnF { ##1 }
6161     {
6162       { NW }
6163       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6164       { NE }
6165       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6166       { SW }
6167       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6168       { SE }
6169       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6170     }
6171     { \@@_error:nn { bad~corner } { ##1 } }
6172   }
```

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

```
6173   \seq_if_empty:NF \l_@@_corners_cells_seq
6174   {
```

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.

```
6175     \tl_gput_right:Nx \g_@@_aux_tl
6176     {
6177       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6178       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6179     }
6180   }
6181 }
```

“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.

```
6182 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6183 {
```

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.

```
6184   \bool_set_false:N \l_tmpa_bool
6185   \int_zero_new:N \l_@@_last_empty_row_int
```



```

6186 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6187 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6188 {
6189   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6190   \bool_lazy_or:nnTF
6191   {
6192     \cs_if_exist_p:c
6193     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6194   }
6195   \l_tmpb_bool
6196   { \bool_set_true:N \l_tmpa_bool }
6197   {
6198     \bool_if:NF \l_tmpa_bool
6199     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6200   }
6201 }

```

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

```

6202 \bool_set_false:N \l_tmpa_bool
6203 \int_zero_new:N \l_@@_last_empty_column_int
6204 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6205 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6206 {
6207   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6208   \bool_lazy_or:nnTF
6209   \l_tmpb_bool
6210   {
6211     \cs_if_exist_p:c
6212     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6213   }
6214   { \bool_set_true:N \l_tmpa_bool }
6215   {
6216     \bool_if:NF \l_tmpa_bool
6217     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6218   }
6219 }

```

Now, we loop over the rows.

```

6220 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6221 {

```

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

```

6222   \bool_set_false:N \l_tmpa_bool
6223   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6224   {
6225     \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6226     \bool_lazy_or:nnTF
6227     \l_tmpb_bool
6228     {
6229       \cs_if_exist_p:c
6230       { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6231     }
6232     { \bool_set_true:N \l_tmpa_bool }
6233     {
6234       \bool_if:NF \l_tmpa_bool
6235       {
6236         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6237         \seq_put_right:Nn
6238         \l_@@_corners_cells_seq
6239         { ##1 - #####1 }
6240       }
6241     }
6242   }
6243 }
6244 }

```

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`).

```

6245 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6246 {
6247   \int_set:Nn \l_tmpa_int { #1 }
6248   \int_set:Nn \l_tmpb_int { #2 }
6249   \bool_set_false:N \l_tmpb_bool
6250   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6251     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6252 }
6253 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6254 {
6255   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6256     {
6257       \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6258         {
6259           \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6260             {
6261               \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6262                 { \bool_set_true:N \l_tmpb_bool }
6263             }
6264         }
6265     }
6266 }
```

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.

```

6267 \bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```

6268 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6269 {
6270   auto-columns-width .code:n =
6271   {
6272     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6273     \dim_gzero_new:N \g_@@_max_cell_width_dim
6274     \bool_set_true:N \l_@@_auto_columns_width_bool
6275   }
6276 }
6277 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6278 {
6279   \int_gincr:N \g_@@_NiceMatrixBlock_int
6280   \dim_zero:N \l_@@_columns_width_dim
6281   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6282   \bool_if:NT \l_@@_block_auto_columns_width_bool
6283     {
6284       \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6285         {
6286           \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
6287             { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6288         }
6289     }
6290 }
```

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`).

```

6291 {
6292   \bool_if:NT \l_@@_block_auto_columns_width_bool
6293   {
6294     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6295     \iow_shipout:Nx \@mainaux
6296     {
6297       \cs_gset:cpn
6298       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6299       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6300     }
6301     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6302   }
6303 }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

25 The extra nodes

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

```

6304 \cs_generate_variant:Nn \dim_min:nn { v n }
6305 \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).

```

6306 \cs_new_protected:Npn \@@_create_extra_nodes:
6307 {
6308   \bool_if:nTF \l_@@_medium_nodes_bool
6309   {
6310     \bool_if:nTF \l_@@_large_nodes_bool
6311     \@@_create_medium_and_large_nodes:
6312     \@@_create_medium_nodes:
6313   }
6314   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6315 }

```

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`.

```

6316 \cs_new_protected:Npn \@@_computations_for_medium_nodes:

```

```

6317 {
6318   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6319   {
6320     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6321     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6322     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6323     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6324   }
6325   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6326   {
6327     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6328     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6329     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6330     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6331   }

```

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

```

6332   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6333   {
6334     \int_step_variable:nnNn
6335     \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.

```

6336     {
6337       \cs_if_exist:cT
6338       { 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`.

```

6339     {
6340       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6341       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6342       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6343       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6344       {
6345         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6346         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6347       }

```

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`.

```

6348       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6349       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6350       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6351       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6352       {
6353         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6354         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6355       }
6356     }
6357   }
6358 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6359   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6360   {
6361     \dim_compare:nNnT
6362     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6363     {
6364       \@@_qpoint:n { row - \@@_i: - base }
6365       \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6366       \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6367     }

```

```

6368     }
6369     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6370     {
6371         \dim_compare:nNnT
6372         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6373         {
6374             \@@_qpoint:n { col - \@@_j: }
6375             \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6376             \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6377         }
6378     }
6379 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6380 \cs_new_protected:Npn \@@_create_medium_nodes:
6381 {
6382     \pgfpicture
6383     \pgfrememberpicturepositiononpagetrue
6384     \pgf@relevantforpicturesizefalse
6385     \@@_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”.

```

6386     \tl_set:Nn \l_@@_suffix_tl { -medium }
6387     \@@_create_nodes:
6388     \endpgfpicture
6389 }

```

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:`.

```

6390 \cs_new_protected:Npn \@@_create_large_nodes:
6391 {
6392     \pgfpicture
6393     \pgfrememberpicturepositiononpagetrue
6394     \pgf@relevantforpicturesizefalse
6395     \@@_computations_for_medium_nodes:
6396     \@@_computations_for_large_nodes:
6397     \tl_set:Nn \l_@@_suffix_tl { - large }
6398     \@@_create_nodes:
6399     \endpgfpicture
6400 }
6401 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6402 {
6403     \pgfpicture
6404     \pgfrememberpicturepositiononpagetrue
6405     \pgf@relevantforpicturesizefalse
6406     \@@_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”.

```

6407     \tl_set:Nn \l_@@_suffix_tl { - medium }
6408     \@@_create_nodes:
6409     \@@_computations_for_large_nodes:
6410     \tl_set:Nn \l_@@_suffix_tl { - large }
6411     \@@_create_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6412 \endpgfpicture
6413 }

```

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.

```

6414 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6415 {
6416   \int_set:Nn \l_@@_first_row_int 1
6417   \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`.

```

6418   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6419   {
6420     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6421     {
6422       (
6423         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6424         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6425       )
6426       / 2
6427     }
6428     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6429     { l_@@_row _ \@@_i: _ min _ dim }
6430   }
6431   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6432   {
6433     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6434     {
6435       (
6436         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6437         \dim_use:c
6438         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6439       )
6440       / 2
6441     }
6442     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6443     { l_@@_column _ \@@_j: _ max _ dim }
6444   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6445   \dim_sub:cn
6446   { l_@@_column _ 1 _ min _ dim }
6447   \l_@@_left_margin_dim
6448   \dim_add:cn
6449   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6450   \l_@@_right_margin_dim
6451 }

```

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).

```

6452 \cs_new_protected:Npn \@@_create_nodes:
6453 {
6454   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6455   {
6456     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6457     {

```

We draw the rectangular node for the cell ($\backslash@@_i$ - $\backslash@@_j$).

```

6458 \@@_pgf_rect_node:nnnnn
6459 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6460 { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6461 { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6462 { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6463 { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6464 \str_if_empty:NF \l_@@_name_str
6465 {
6466 \pgfnodealias
6467 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6468 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6469 }
6470 }
6471 }

```

Now, we create the nodes for the cells of the $\backslash\text{multicolumn}$. We recall that we have stored in $\backslash\text{g_}@@_multicolumn_cells_seq$ the list of the cells where a $\backslash\text{multicolumn}\{n\}\{\dots\}\{\dots\}$ with $n>1$ was issued and in $\backslash\text{g_}@@_multicolumn_sizes_seq$ the correspondent values of n .

```

6472 \cs_if_exist_use:NF
6473 \seq_map_pairwise_function:NNN
6474 \seq_mapthread_function:NNN
6475 \g_@@_multicolumn_cells_seq
6476 \g_@@_multicolumn_sizes_seq
6477 \@@_node_for_multicolumn:nn
6478 }

```

```

6479 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6480 {
6481 \cs_set_nopar:Npn \@@_i: { #1 }
6482 \cs_set_nopar:Npn \@@_j: { #2 }
6483 }

```

The command $\backslash\text{@@_node_for_multicolumn:nn}$ takes two arguments. The first is the position of the cell where the command $\backslash\text{multicolumn}\{n\}\{\dots\}\{\dots\}$ was issued in the format i - j and the second is the value of n (the length of the “multi-cell”).

```

6484 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6485 {
6486 \@@_extract_coords_values: #1 \q_stop
6487 \@@_pgf_rect_node:nnnnn
6488 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6489 { \dim_use:c { l_@@_column_ \@@_j: _min _ dim } }
6490 { \dim_use:c { l_@@_row_ \@@_i: _min _ dim } }
6491 { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6492 { \dim_use:c { l_@@_row_ \@@_i: _max _ dim } }
6493 \str_if_empty:NF \l_@@_name_str
6494 {
6495 \pgfnodealias
6496 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6497 { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6498 }
6499 }

```

26 The blocks

The code deals with the command $\backslash\text{Block}$. This command has no direct link with the environment $\{\text{NiceMatrixBlock}\}$.

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.

```

6500 \keys_define:nn { NiceMatrix / Block / FirstPass }
6501 {
6502   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6503   l .value_forbidden:n = true ,
6504   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6505   r .value_forbidden:n = true ,
6506   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6507   c .value_forbidden:n = true ,
6508   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6509   L .value_forbidden:n = true ,
6510   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6511   R .value_forbidden:n = true ,
6512   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6513   C .value_forbidden:n = true ,
6514   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6515   t .value_forbidden:n = true ,
6516   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6517   b .value_forbidden:n = true ,
6518   color .tl_set:N = \l_@@_color_tl ,
6519   color .value_required:n = true ,
6520   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6521   respect-arraystretch .default:n = true ,
6522 }

```

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.

```

6523 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
6524 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not be provided by the user, you use 1-1 (that is to say a block of only one cell).

```

6525   \peek_remove_spaces:n
6526   {
6527     \tl_if_blank:nTF { #2 }
6528     { \@@_Block_i 1-1 \q_stop }
6529     {
6530       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6531       \@@_Block_i_czech \@@_Block_i
6532       #2 \q_stop
6533     }
6534     { #1 } { #3 } { #4 }
6535   }
6536 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6537 \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`.

```

6538 {
6539   \char_set_catcode_active:N -
6540   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6541 }

```

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.

```

6542 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6543 {

```


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).

```

6544 \bool_lazy_or:nnTF
6545   { \tl_if_blank_p:n { #1 } }
6546   { \str_if_eq_p:nn { #1 } { * } }
6547   { \int_set:Nn \l_tmpa_int { 100 } }
6548   { \int_set:Nn \l_tmpa_int { #1 } }
6549 \bool_lazy_or:nnTF
6550   { \tl_if_blank_p:n { #2 } }
6551   { \str_if_eq_p:nn { #2 } { * } }
6552   { \int_set:Nn \l_tmpb_int { 100 } }
6553   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6554 \int_compare:nNnTF \l_tmpb_int = 1
6555 {
6556   \str_if_empty:NTF \l_@@_hpos_cell_str
6557     { \str_set:Nn \l_@@_hpos_block_str c }
6558     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6559 }
6560 { \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.

```

6561 \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
6562 \tl_set:Nx \l_tmpa_tl
6563 {
6564   { \int_use:N \c@iRow }
6565   { \int_use:N \c@jCol }
6566   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6567   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6568 }

```

Now, \l_tmpa_tl contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

$\{imin\}\{jmin\}\{imax\}\{jmax\}$.

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: \@@_Block_iv:nnnnn and \@@_Block_v:nnnnn (the five arguments of those macros are provided by currying).

```

6569 \bool_if:nTF
6570 {
6571   (
6572     \int_compare_p:nNn { \l_tmpa_int } = 1
6573     ||
6574     \int_compare_p:nNn { \l_tmpb_int } = 1
6575   )
6576   && ! \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.

```

6577   && ! \l_@@_X_column_bool
6578 }
6579 { \exp_args:Nxx \@@_Block_iv:nnnnn }
6580 { \exp_args:Nxx \@@_Block_v:nnnnn }
6581 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6582 }

```

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).

```

6583 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6584 {
6585   \int_gincr:N \g_@@_block_box_int
6586   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6587   {
6588     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6589     {
6590       \@@_actually_diagbox:nnnnnn
6591       { \int_use:N \c@iRow }
6592       { \int_use:N \c@jCol }
6593       { \int_eval:n { \c@iRow + #1 - 1 } }
6594       { \int_eval:n { \c@jCol + #2 - 1 } }
6595       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6596     }
6597   }
6598   \box_gclear_new:c
6599   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
6600   \hbox_gset:cn
6601   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
6602   {

```

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}`).

```

6603   \tl_if_empty:NTF \l_@@_color_tl
6604   { \int_compare:nNnT { #2 } = 1 \set@color }
6605   { \@@_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`.

```

6606   \int_compare:nNnT { #1 } = 1
6607   {
6608     \int_compare:nNnTF \c@iRow = 0
6609     \l_@@_code_for_first_row_tl
6610     {
6611       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6612       \l_@@_code_for_last_row_tl
6613     }
6614     \g_@@_row_style_tl
6615   }
6616   \group_begin:
6617   \bool_if:NF \l_@@_respect_arraystretch_bool
6618   { \cs_set:Npn \arraystretch { 1 } }
6619   \dim_zero:N \extrarowheight
6620   #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).

```

6621   \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
6622   \bool_if:NTF \l_@@_NiceTabular_bool
6623   {
6624     \bool_lazy_all:nTF
6625     {

```

```

6626         { \int_compare_p:nNn { #2 } = 1 }
6627         { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6628         { ! \g_@@_rotate_bool }
6629     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

6630     {
6631         \use:x
6632         {
6633             \exp_not:N \begin { minipage }%
6634             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6635             { \l_@@_col_width_dim }
6636             \str_case:Vn \l_@@_hpos_block_str
6637             {
6638                 c \centering
6639                 r \raggedleft
6640                 l \raggedright
6641             }
6642         }
6643         #5
6644     \end { minipage }
6645     }
6646     {
6647         \use:x
6648         {
6649             \exp_not:N \begin { tabular }%
6650             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6651             { @ { } \l_@@_hpos_block_str @ { } }
6652         }
6653         #5
6654     \end { tabular }
6655     }
6656 }
6657 {
6658     \c_math_toggle_token
6659     \use:x
6660     {
6661         \exp_not:N \begin { array }%
6662         [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6663         { @ { } \l_@@_hpos_block_str @ { } }
6664     }
6665     #5
6666     \end { array }
6667     \c_math_toggle_token
6668 }
6669 \group_end:
6670 }
6671 \bool_if:NT \g_@@_rotate_bool
6672 {
6673     \box_grotate:cn
6674     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6675     { 90 }
6676     \bool_gset_false:N \g_@@_rotate_bool
6677 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

6678     \int_compare:nNnT { #2 } = 1
6679     {
6680         \dim_gset:Nn \g_@@_blocks_wd_dim
6681         {
6682             \dim_max:nn
6683             \g_@@_blocks_wd_dim
6684             {

```

```

6685         \box_wd:c
6686         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6687     }
6688 }
6689 }

```

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.

```

6690 \int_compare:nNnT { #1 } = 1
6691 {
6692     \dim_gset:Nn \g_@@_blocks_ht_dim
6693     {
6694         \dim_max:nn
6695         \g_@@_blocks_ht_dim
6696         {
6697             \box_ht:c
6698             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6699         }
6700     }
6701     \dim_gset:Nn \g_@@_blocks_dp_dim
6702     {
6703         \dim_max:nn
6704         \g_@@_blocks_dp_dim
6705         {
6706             \box_dp:c
6707             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6708         }
6709     }
6710 }
6711 \seq_gput_right:Nx \g_@@_blocks_seq
6712 {
6713     \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.

```

6714     { \exp_not:n { #3 } , \l_@@_hpos_block_str }
6715     {
6716         \box_use_drop:c
6717         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6718     }
6719 }
6720 }

```

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.

```

6721 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
6722 {
6723     \seq_gput_right:Nx \g_@@_blocks_seq
6724     {
6725         \l_tmpa_tl
6726         { \exp_not:n { #3 } }
6727         {
6728             \bool_if:NTF \l_@@_NiceTabular_bool
6729             {
6730                 \group_begin:
6731                 \bool_if:NF \l_@@_respect_arraystretch_bool
6732                 { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6733                 \exp_not:n
6734                 {
6735                     \dim_zero:N \extrarowheight
6736                     #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).

```

6737         \bool_if:NT \g_@@_rotate_bool
6738         { \str_set:Nn \l_@@_hpos_block_str c }
6739         \use:x
6740         {
6741             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
6742             { @ { } \l_@@_hpos_block_str @ { } }
6743         }
6744         #5
6745         \end { tabular }
6746     }
6747     \group_end:
6748 }
6749 {
6750     \group_begin:
6751     \bool_if:NF \l_@@_respect_arraystretch_bool
6752     { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6753     \exp_not:n
6754     {
6755         \dim_zero:N \extrarowheight
6756         #4
6757         \bool_if:NT \g_@@_rotate_bool
6758         { \str_set:Nn \l_@@_hpos_block_str c }
6759         \c_math_toggle_token
6760         \use:x
6761         {
6762             \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
6763             { @ { } \l_@@_hpos_block_str @ { } }
6764         }
6765         #5
6766         \end { array }
6767         \c_math_toggle_token
6768     }
6769     \group_end:
6770 }
6771 }
6772 }
6773 }
```

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).

```

6774 \keys_define:nn { NiceMatrix / Block / SecondPass }
6775 {
6776     tikz .code:n =
6777         \IfPackageLoadedTF { tikz }
6778         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6779         { \@@_error:n { tikz-key-without~tikz } } ,
6780     tikz .value_required:n = true ,
6781     fill .code:n =
6782         \tl_set_rescan:Nnn
6783         \l_@@_fill_tl
6784         { \char_set_catcode_other:N ! }
6785         { #1 } ,
6786     fill .value_required:n = true ,
6787     draw .code:n =
6788         \tl_set_rescan:Nnn
6789         \l_@@_draw_tl
```

```

6790     { \char_set_catcode_other:N ! }
6791     { #1 } ,
6792     draw .default:n = default ,
6793     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6794     rounded-corners .default:n = 4 pt ,
6795     color .code:n =
6796       \@@_color:n { #1 }
6797       \tl_set_rescan:Nnn
6798         \l_@@_draw_tl
6799         { \char_set_catcode_other:N ! }
6800         { #1 } ,
6801     color .value_required:n = true ,
6802     borders .clist_set:N = \l_@@_borders_clist ,
6803     borders .value_required:n = true ,
6804     hvlines .meta:n = { vlines , hlines } ,
6805     vlines .bool_set:N = \l_@@_vlines_block_bool ,
6806     vlines .default:n = true ,
6807     hlines .bool_set:N = \l_@@_hlines_block_bool ,
6808     hlines .default:n = true ,
6809     line-width .dim_set:N = \l_@@_line_width_dim ,
6810     line-width .value_required:n = true ,
6811     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6812     l .value_forbidden:n = true ,
6813     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6814     r .value_forbidden:n = true ,
6815     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6816     c .value_forbidden:n = true ,
6817     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6818       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6819     L .value_forbidden:n = true ,
6820     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6821       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6822     R .value_forbidden:n = true ,
6823     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6824       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6825     C .value_forbidden:n = true ,
6826     t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6827     t .value_forbidden:n = true ,
6828     T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6829     T .value_forbidden:n = true ,
6830     b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6831     b .value_forbidden:n = true ,
6832     B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6833     B .value_forbidden:n = true ,
6834     v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
6835     v-center .value_forbidden:n = true ,
6836     name .tl_set:N = \l_@@_block_name_str ,
6837     name .value_required:n = true ,
6838     name .initial:n = ,
6839     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6840     respect-arraystretch .default:n = true ,
6841     transparent .bool_set:N = \l_@@_transparent_bool ,
6842     transparent .default:n = true ,
6843     transparent .initial:n = false ,
6844     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
6845   }

```

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.

```

6846 \cs_new_protected:Npn \@@_draw_blocks:
6847 {
6848   \cs_set_eq:NN \ialign \@@_old_ialign:

```

```

6849 \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
6850 }
6851 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6852 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6853 \int_zero_new:N \l_@@_last_row_int
6854 \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.

```

6855 \int_compare:nNnTF { #3 } > { 99 }
6856 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
6857 { \int_set:Nn \l_@@_last_row_int { #3 } }
6858 \int_compare:nNnTF { #4 } > { 99 }
6859 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
6860 { \int_set:Nn \l_@@_last_col_int { #4 } }
6861 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6862 {
6863   \int_compare:nTF
6864   { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6865   {
6866     \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
6867     \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
6868     \@@_msg_redirect_name:nn { columns-not-used } { none }
6869   }
6870   { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6871 }
6872 {
6873   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6874   { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6875   { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6876 }
6877 }

```

`#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

```

6878 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6879 {

```

The group is for the keys.

```

6880 \group_begin:
6881 \int_compare:nNnT { #1 } = { #3 }
6882 { \str_set:Nn \l_@@_vpos_of_block_str { t } }
6883 \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
6884 \bool_if:NT \l_@@_vlines_block_bool
6885 {
6886   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6887   {
6888     \@@_vlines_block:nnn
6889     { \exp_not:n { #5 } }
6890     { #1 - #2 }
6891     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6892   }
6893 }
6894 \bool_if:NT \l_@@_hlines_block_bool
6895 {
6896   \tl_gput_right:Nx \g_nicematrix_code_after_tl

```

```

6897     {
6898         \@@_hlines_block:nnn
6899         { \exp_not:n { #5 } }
6900         { #1 - #2 }
6901         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6902     }
6903 }
6904 \bool_if:nF
6905 {
6906     \l_@@_transparent_bool
6907     || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
6908 }
6909 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

6910     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6911     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6912 }

6913 \bool_lazy_and:nnT
6914 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
6915 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
6916 { \@@_error:n { hlines-with-color } }

6917 \tl_if_empty:NF \l_@@_draw_tl
6918 {
6919     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6920     {
6921         \@@_stroke_block:nnn
6922         { \exp_not:n { #5 } }
6923         { #1 - #2 }
6924         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6925     }
6926     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6927     { { #1 } { #2 } { #3 } { #4 } }
6928 }

6929 \clist_if_empty:NF \l_@@_borders_clist
6930 {
6931     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6932     {
6933         \@@_stroke_borders_block:nnn
6934         { \exp_not:n { #5 } }
6935         { #1 - #2 }
6936         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6937     }
6938 }

6939 \tl_if_empty:NF \l_@@_fill_tl
6940 {
6941     \tl_gput_right:Nx \g_@@_pre_code_before_tl
6942     {
6943         \exp_not:N \roundedrectanglecolor
6944         \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
6945             { \l_@@_fill_tl }
6946             { { \l_@@_fill_tl } }
6947             { #1 - #2 }
6948             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6949             { \dim_use:N \l_@@_rounded_corners_dim }
6950         ]
6951     }

```



```

6952 \seq_if_empty:NF \l_@@_tikz_seq
6953 {
6954   \tl_gput_right:Nx \g_nicematrix_code_before_tl
6955   {
6956     \@@_block_tikz:nnnnn
6957     { #1 }
6958     { #2 }
6959     { \int_use:N \l_@@_last_row_int }
6960     { \int_use:N \l_@@_last_col_int }
6961     { \seq_use:Nn \l_@@_tikz_seq { , } }
6962   }
6963 }

6964 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6965 {
6966   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6967   {
6968     \@@_actually_diagbox:nnnnnn
6969     { #1 }
6970     { #2 }
6971     { \int_use:N \l_@@_last_row_int }
6972     { \int_use:N \l_@@_last_col_int }
6973     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6974   }
6975 }

6976 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6977 \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.

```

6978 \pgfpicture
6979 \pgfrememberpicturepositiononpagetrue
6980 \pgf@relevantforpicturesizefalse
6981 \@@_qpoint:n { row - #1 }
6982 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6983 \@@_qpoint:n { col - #2 }
6984 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6985 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6986 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6987 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6988 \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.

```

6989 \@@pgf_rect_node:nnnnn
6990 { \@@_env: - #1 - #2 - block }
6991 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6992 \str_if_empty:NF \l_@@_block_name_str
6993 {
6994   \pgfnodealias
6995   { \@@_env: - \l_@@_block_name_str }
6996   { \@@_env: - #1 - #2 - block }
6997   \str_if_empty:NF \l_@@_name_str
6998   {
6999     \pgfnodealias
7000     { \l_@@_name_str - \l_@@_block_name_str }
7001     { \@@_env: - #1 - #2 - block }
7002   }
7003 }

```

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.

```

7004 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7005 {
7006   \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.

```

7007 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7008 {

```

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.

```

7009 \cs_if_exist:cT
7010 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7011 {
7012   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7013   {
7014     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7015     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7016   }
7017 }
7018 }

```

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.

```

7019 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7020 {
7021   \@@_qpoint:n { col - #2 }
7022   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7023 }
7024 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7025 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7026 {
7027   \cs_if_exist:cT
7028   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7029   {
7030     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7031     {
7032       \pgfpointanchor
7033       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7034       { east }

```

```

7035         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7036     }
7037 }
7038 }
7039 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7040 {
7041     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7042     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7043 }
7044 \@@_pgf_rect_node:nnnnn
7045 { \@@_env: - #1 - #2 - block - short }
7046 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7047 }

```

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.

```

7048 \bool_if:NT \l_@@_medium_nodes_bool
7049 {
7050     \@@_pgf_rect_node:nnn
7051     { \@@_env: - #1 - #2 - block - medium }
7052     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7053     {
7054         \pgfpointanchor
7055         { \@@_env:
7056             - \int_use:N \l_@@_last_row_int
7057             - \int_use:N \l_@@_last_col_int - medium
7058         }
7059         { south-east }
7060     }
7061 }

```

Now, we will put the label of the block.

```

7062 \bool_lazy_any:nTF
7063 {
7064     { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7065     { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7066     { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7067 }
7068 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7069 \int_compare:nNnT { #2 } = 0
7070 { \str_set:Nn \l_@@_hpos_block_str r }
7071 \bool_if:nT \g_@@_last_col_found_bool
7072 {
7073     \int_compare:nNnT { #2 } = \g_@@_col_total_int
7074     { \str_set:Nn \l_@@_hpos_block_str l }
7075 }
7076 \tl_set:Nx \l_tmpa_tl
7077 {
7078     \str_case:Vn \l_@@_vpos_of_block_str
7079     {
7080         c {
7081             \str_case:Vn \l_@@_hpos_block_str
7082             {
7083                 c { center }
7084                 l { west }
7085                 r { east }
7086             }
7087         }
7088     }
7089     T {

```

```

7090         \str_case:Vn \l_@@_hpos_block_str
7091         {
7092             c { north }
7093             l { north-west }
7094             r { north-east }
7095         }
7096
7097     }
7098     B {
7099         \str_case:Vn \l_@@_hpos_block_str
7100         {
7101             c { south }
7102             l { south-west }
7103             r { south-east }
7104         }
7105
7106     }
7107 }
7108
7109 \pgftransformshift
7110 {
7111     \pgfpointanchor
7112     {
7113         \@@_env: - #1 - #2 - block
7114         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7115     }
7116     { \l_tmpa_tl }
7117 }
7118 \pgfset
7119 {
7120     inner-xsep = \c_zero_dim ,
7121     inner-ysep = \l_@@_block_ysep_dim
7122 }
7123 \pgfnode
7124 { rectangle }
7125 { \l_tmpa_tl }
7126 { \box_use_drop:N \l_@@_cell_box } { } { } { }
7127 }
7128 {
7129     \pgfextracty \l_tmpa_dim
7130     {
7131         \@@_qpoint:n
7132         {
7133             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7134             - base
7135         }
7136     }
7137     \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.

```

7138 \pgfpointanchor
7139 {
7140     \@@_env: - #1 - #2 - block
7141     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7142 }
7143 {
7144     \str_case:Vn \l_@@_hpos_block_str
7145     {
7146         c { center }
7147         l { west }
7148         r { east }
7149     }
7150 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7151     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7152     \pgfset { inner~sep = \c_zero_dim }
7153     \pgfnode
7154     { rectangle }
7155     {
7156         \str_case:Vn \l_@@_hpos_block_str
7157         {
7158             c { base }
7159             l { base~west }
7160             r { base~east }
7161         }
7162     }
7163     { \box_use_drop:N \l_@@_cell_box } { } { }
7164 }
7165 \endpgfpicture
7166 \group_end:
7167 }

```

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).

```

7168 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7169 {
7170     \group_begin:
7171     \tl_clear:N \l_@@_draw_tl
7172     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7173     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7174     \pgfpicture
7175     \pgfrememberpicturepositiononpagetrue
7176     \pgf@relevantforpicturesizefalse
7177     \tl_if_empty:NF \l_@@_draw_tl
7178     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7179         \str_if_eq:VnTF \l_@@_draw_tl { default }
7180         { \CT@arc@ }
7181         { \@@_color:V \l_@@_draw_tl }
7182     }
7183     \pgfsetcornersarced
7184     {
7185         \pgfpoint
7186         { \dim_use:N \l_@@_rounded_corners_dim }
7187         { \dim_use:N \l_@@_rounded_corners_dim }
7188     }
7189     \@@_cut_on_hyphen:w #2 \q_stop
7190     \bool_lazy_and:nnT
7191     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7192     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7193     {
7194         \@@_qpoint:n { row - \l_tmpa_tl }
7195         \dim_set:Nn \l_tmpb_dim { \pgf@y }
7196         \@@_qpoint:n { col - \l_tmpb_tl }
7197         \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
7198         \@@_cut_on_hyphen:w #3 \q_stop
7199         \int_compare:nNnT \l_tmpa_tl > \c@iRow
7200         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7201         \int_compare:nNnT \l_tmpb_tl > \c@jCol
7202         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7203         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7204         \dim_set:Nn \l_tmpa_dim { \pgf@y }

```

```

7205 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7206 \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
7207 \pgfpathrectanglecorners
7208 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7209 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7210 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7211 \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7212 { \pgfusepathqstroke }
7213 { \pgfusepath { stroke } }
7214 }
7215 \endpgfpicture
7216 \group_end:
7217 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7218 \keys_define:nn { NiceMatrix / BlockStroke }
7219 {
7220   color .tl_set:N = \l_@@_draw_tl ,
7221   draw .tl_set:N = \l_@@_draw_tl ,
7222   draw .default:n = default ,
7223   line-width .dim_set:N = \l_@@_line_width_dim ,
7224   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7225   rounded-corners .default:n = 4 pt
7226 }

```

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).

```

7227 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7228 {
7229   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7230   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7231   \@@_cut_on_hyphen:w #2 \q_stop
7232   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7233   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7234   \@@_cut_on_hyphen:w #3 \q_stop
7235   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7236   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7237   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7238   {
7239     \use:x
7240     {
7241       \@@_vline:n
7242       {
7243         position = ##1 ,
7244         start = \l_@@_tmpc_tl ,
7245         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7246         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7247       }
7248     }
7249   }
7250 }
7251 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7252 {
7253   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7254   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7255   \@@_cut_on_hyphen:w #2 \q_stop
7256   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7257   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7258   \@@_cut_on_hyphen:w #3 \q_stop
7259   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7260   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7261   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl

```

```

7262 {
7263   \use:x
7264   {
7265     \@@_hline:n
7266     {
7267       position = ##1 ,
7268       start = \l_@@_tmpd_tl ,
7269       end = \int_eval:n { \l_tmpb_tl - 1 } ,
7270       total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7271     }
7272   }
7273 }
7274 }

```

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).

```

7275 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7276 {
7277   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7278   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7279   \dim_compare:NnTF \l_@@_rounded_corners_dim > \c_zero_dim
7280     { \@@_error:n { borders~forbidden } }
7281     {
7282       \tl_clear_new:N \l_@@_borders_tikz_tl
7283       \keys_set:nV
7284         { NiceMatrix / OnlyForTikzInBorders }
7285       \l_@@_borders_clist
7286       \@@_cut_on_hyphen:w #2 \q_stop
7287       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7288       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7289       \@@_cut_on_hyphen:w #3 \q_stop
7290       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7291       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7292       \@@_stroke_borders_block_i:
7293     }
7294 }
7295 \hook_gput_code:nnn { begindocument } { . }
7296 {
7297   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7298   {
7299     \c_@@_pgfortikzpicture_tl
7300     \@@_stroke_borders_block_ii:
7301     \c_@@_endpgfortikzpicture_tl
7302   }
7303 }
7304 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7305 {
7306   \pgfrememberpicturepositiononpagetrue
7307   \pgf@relevantforpicturesizefalse
7308   \CT@arc@
7309   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7310   \clist_if_in:NnT \l_@@_borders_clist { right }
7311     { \@@_stroke_vertical:n \l_tmpb_tl }
7312   \clist_if_in:NnT \l_@@_borders_clist { left }
7313     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7314   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7315     { \@@_stroke_horizontal:n \l_tmpa_tl }
7316   \clist_if_in:NnT \l_@@_borders_clist { top }
7317     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7318 }

```

```

7319 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7320 {
7321   tikz .code:n =
7322     \cs_if_exist:NTF \tikzpicture
7323     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7324     { \@@_error:n { tikz-in-borders-without-tikz } } ,
7325   tikz .value_required:n = true ,
7326   top .code:n = ,
7327   bottom .code:n = ,
7328   left .code:n = ,
7329   right .code:n = ,
7330   unknown .code:n = \@@_error:n { bad-border }
7331 }

```

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).

```

7332 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7333 {
7334   \@@_qpoint:n \l_@@_tmpc_tl
7335   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7336   \@@_qpoint:n \l_tmpa_tl
7337   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7338   \@@_qpoint:n { #1 }
7339   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7340   {
7341     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7342     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7343     \pgfusepathqstroke
7344   }
7345   {
7346     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7347     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7348   }
7349 }

```

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).

```

7350 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7351 {
7352   \@@_qpoint:n \l_@@_tmpd_tl
7353   \clist_if_in:NnTF \l_@@_borders_clist { left }
7354   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7355   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7356   \@@_qpoint:n \l_tmpb_tl
7357   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7358   \@@_qpoint:n { #1 }
7359   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7360   {
7361     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7362     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7363     \pgfusepathqstroke
7364   }
7365   {
7366     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7367     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7368   }
7369 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

7370 \keys_define:nn { NiceMatrix / BlockBorders }
7371 {
7372   borders .clist_set:N = \l_@@_borders_clist ,

```



```

7373   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7374   rounded-corners .default:n = 4 pt ,
7375   line-width .dim_set:N = \l_@@_line_width_dim ,
7376 }

```

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.

```

7377 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7378 {
7379   \begin { tikzpicture }
7380   \clist_map_inline:nn { #5 }
7381   {
7382     \path [ ##1 ]
7383           ( #1 -| #2 )
7384           rectangle
7385           ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7386   }
7387   \end { tikzpicture }
7388 }

```

27 How to draw the dotted lines transparently

```

7389 \cs_set_protected:Npn \@@_renew_matrix:
7390 {
7391   \RenewDocumentEnvironment { pmatrix } { } {
7392     { \pNiceMatrix }
7393     { \endpNiceMatrix }
7394   }
7395   \RenewDocumentEnvironment { vmatrix } { } {
7396     { \vNiceMatrix }
7397     { \endvNiceMatrix }
7398   }
7399   \RenewDocumentEnvironment { Vmatrix } { } {
7400     { \VNiceMatrix }
7401     { \endVNiceMatrix }
7402   }
7403   \RenewDocumentEnvironment { bmatrix } { } {
7404     { \bNiceMatrix }
7405     { \endbNiceMatrix }
7406   }
7407   \RenewDocumentEnvironment { Bmatrix } { } {
7408     { \BNiceMatrix }
7409     { \endBNiceMatrix }
7410   }
7411 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7407 \keys_define:nn { NiceMatrix / Auto }
7408 {
7409   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7410   columns-type .value_required:n = true ,
7411   l .meta:n = { columns-type = l } ,
7412   r .meta:n = { columns-type = r } ,
7413   c .meta:n = { columns-type = c } ,
7414   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7415   delimiters / color .value_required:n = true ,
7416   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7417   delimiters / max-width .default:n = true ,
7418   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7419   delimiters .value_required:n = true ,

```

```

7420 }
7421 \NewDocumentCommand \AutoNiceMatrixWithDelims
7422 { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7423 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7424 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7425 {

```

The group is for the protection of the keys.

```

7426 \group_begin:
7427 \bool_set_true:N \l_@@_Matrix_bool
7428 \str_if_eq:nnT { #1 } { . }
7429 {
7430 \str_if_eq:nnT { #2 } { . }
7431 { \bool_set_false:N \l_@@_Matrix_bool }
7432 }
7433 \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl

```

We nullify the command `\@@_transform_preamble:` because we will provide a preamble which is yet transformed (by using `\l_@@_columns_type_tl` which is yet nicematrix-ready).

```

7434 \cs_set_eq:NN \@@_transform_preamble: \prg_do_nothing:
7435 \use:x
7436 {
7437 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7438 { * { #4 } { \exp_not:N \l_@@_columns_type_tl } }
7439 [ \exp_not:N \l_tmpa_tl ]
7440 }
7441 \int_compare:nNnT \l_@@_first_row_int = 0
7442 {
7443 \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7444 \prg_replicate:nn { #4 - 1 } { & }
7445 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7446 }
7447 \prg_replicate:nn { #3 }
7448 {
7449 \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`).

```

7450 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7451 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7452 }
7453 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7454 {
7455 \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7456 \prg_replicate:nn { #4 - 1 } { & }
7457 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7458 }
7459 \end { NiceArrayWithDelims }
7460 \group_end:
7461 }
7462 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7463 {
7464 \cs_set_protected:cpn { #1 AutoNiceMatrix }
7465 {
7466 \bool_gset_false:N \g_@@_NiceArray_bool
7467 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7468 \AutoNiceMatrixWithDelims { #2 } { #3 }
7469 }
7470 }

```

```

7471 \@@_define_com:nnn p ( )
7472 \@@_define_com:nnn b [ ]
7473 \@@_define_com:nnn v | |
7474 \@@_define_com:nnn V \| \|
7475 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7476 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
7477 {
7478   \group_begin:
7479   \bool_gset_true:N \g_@@_NiceArray_bool
7480   % \bool_set_true:N \l_@@_Matrix_bool
7481   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7482   \group_end:
7483 }

```

29 The redefinition of the command `\dotfill`

```

7484 \cs_set_eq:NN \@@_old_dotfill \dotfill
7485 \cs_new_protected:Npn \@@_dotfill:
7486 {

```

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}`).

```

7487   \@@_old_dotfill
7488   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7489 }

```

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`.

```

7490 \cs_new_protected:Npn \@@_dotfill_i:
7491 { \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.

```

7492 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7493 {
7494   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7495   {
7496     \@@_actually_diagbox:nnnnnn
7497     { \int_use:N \c@iRow }
7498     { \int_use:N \c@jCol }
7499     { \int_use:N \c@iRow }
7500     { \int_use:N \c@jCol }
7501     { \exp_not:n { #1 } }
7502     { \exp_not:n { #2 } }
7503   }

```

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`.

```

7504   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7505   {
7506     { \int_use:N \c@iRow }
7507     { \int_use:N \c@jCol }
7508     { \int_use:N \c@iRow }
7509     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7510         { }
7511     }
7512 }

```

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.

```

7513 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7514 {
7515     \pgfpicture
7516     \pgf@relevantforpicturesizefalse
7517     \pgfrememberpicturepositiononpagetrue
7518     \@@_qpoint:n { row - #1 }
7519     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7520     \@@_qpoint:n { col - #2 }
7521     \dim_set_eq:NN \l_tmpb_dim \pgf@x
7522     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7523     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7524     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7525     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7526     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7527     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7528     {

```

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.

```

7529         \CT@arc@
7530         \pgfsetroundcap
7531         \pgfusepathqstroke
7532     }
7533     \pgfset { inner~sep = 1 pt }
7534     \pgfscope
7535     \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7536     \pgfnode { rectangle } { south~west }
7537     {
7538         \begin { minipage } { 20 cm }
7539         \@@_math_toggle_token: #5 \@@_math_toggle_token:
7540         \end { minipage }
7541     }
7542     { }
7543     { }
7544 \endpgfscope
7545 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7546 \pgfnode { rectangle } { north~east }
7547 {
7548     \begin { minipage } { 20 cm }
7549     \raggedleft
7550     \@@_math_toggle_token: #6 \@@_math_toggle_token:
7551     \end { minipage }
7552 }
7553 { }
7554 { }
7555 \endpgfpicture
7556 }

```

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.

```

7557 \keys_define:nn { NiceMatrix }
7558 {
7559   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7560   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7561 }
7562 \keys_define:nn { NiceMatrix / CodeAfter }
7563 {
7564   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7565   sub-matrix .value_required:n = true ,
7566   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7567   delimiters / color .value_required:n = true ,
7568   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7569   rules .value_required:n = true ,
7570   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
7571 }
```

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`.

```

7572 \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 `\`.

```

7573 \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`.

```

7574 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7575 {
7576   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7577   \@@_CodeAfter_iv:n
7578 }
```

We catch the argument of the command `\end` (in `#1`).

```

7579 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7580 {
```

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.

```

7581   \str_if_eq:eeTF \@currentenv { #1 }
7582   { \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`.

```

7583   {
7584     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7585     \@@_CodeAfter_ii:n
7586   }
7587 }
```

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_t1` (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.

```

7588 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7589 {
7590   \pgfpicture
7591   \pgfrememberpicturepositiononpagetrue
7592   \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.

```

7593   \@@_qpoint:n { row - 1 }
7594   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7595   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7596   \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).

```

7597   \bool_if:nTF { #3 }
7598   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7599   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7600   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7601   {
7602     \cs_if_exist:cT
7603     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7604     {
7605       \pgfpointanchor
7606       { \@@_env: - ##1 - #2 }
7607       { \bool_if:nTF { #3 } { west } { east } }
7608       \dim_set:Nn \l_tmpa_dim
7609       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7610     }
7611   }

```

Now we can put the delimiter with a node of PGF.

```

7612   \pgfset { inner~sep = \c_zero_dim }
7613   \dim_zero:N \nulldelimiterspace
7614   \pgftransformshift
7615   {
7616     \pgfpoint
7617     { \l_tmpa_dim }
7618     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7619   }
7620   \pgfnode
7621   { rectangle }
7622   { \bool_if:nTF { #3 } { east } { west } }
7623   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7624     \nullfont
7625     \c_math_toggle_token
7626     \@@_color:V \l_@@_delimiters_color_t1
7627     \bool_if:nTF { #3 } { \left #1 } { \left . }

```

```

7628     \vcenter
7629     {
7630         \nullfont
7631         \hrule \@height
7632             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7633             \@depth \c_zero_dim
7634             \@width \c_zero_dim
7635     }
7636     \bool_if:nTF { #3 } { \right . } { \right #1 }
7637     \c_math_toggle_token
7638 }
7639 { }
7640 { }
7641 \endpgfpicture
7642 }

```

33 The command \SubMatrix

```

7643 \keys_define:nn { NiceMatrix / sub-matrix }
7644 {
7645     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7646     extra-height .value_required:n = true ,
7647     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7648     left-xshift .value_required:n = true ,
7649     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7650     right-xshift .value_required:n = true ,
7651     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7652     xshift .value_required:n = true ,
7653     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7654     delimiters / color .value_required:n = true ,
7655     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7656     slim .default:n = true ,
7657     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7658     hlines .default:n = all ,
7659     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7660     vlines .default:n = all ,
7661     hvlines .meta:n = { hlines, vlines } ,
7662     hvlines .value_forbidden:n = true ,
7663 }
7664 \keys_define:nn { NiceMatrix }
7665 {
7666     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7667     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7668     NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7669     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7670     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7671     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7672 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

7673 \keys_define:nn { NiceMatrix / SubMatrix }
7674 {
7675     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7676     delimiters / color .value_required:n = true ,
7677     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7678     hlines .default:n = all ,
7679     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7680     vlines .default:n = all ,
7681     hvlines .meta:n = { hlines, vlines } ,

```

```

7682   hvlines .value_forbidden:n = true ,
7683   name .code:n =
7684     \tl_if_empty:nTF { #1 }
7685     { \@@_error:n { Invalid~name } }
7686     {
7687       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7688       {
7689         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7690         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
7691         {
7692           \str_set:Nn \l_@@_submatrix_name_str { #1 }
7693           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7694         }
7695       }
7696       { \@@_error:n { Invalid~name } }
7697     } ,
7698   name .value_required:n = true ,
7699   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7700   rules .value_required:n = true ,
7701   code .tl_set:N = \l_@@_code_tl ,
7702   code .value_required:n = true ,
7703   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
7704 }

7705 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7706 {
7707   \peek_remove_spaces:n
7708   {
7709     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7710     {
7711       \SubMatrix { #1 } { #2 } { #3 } { #4 }
7712       [
7713         delimiters / color = \l_@@_delimiters_color_tl ,
7714         hlines = \l_@@_submatrix_hlines_clist ,
7715         vlines = \l_@@_submatrix_vlines_clist ,
7716         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
7717         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
7718         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
7719         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
7720         #5
7721       ]
7722     }
7723     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7724   }
7725 }

7726 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
7727 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7728 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

7729 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
7730 {
7731   \seq_gput_right:Nx \g_@@_submatrix_seq
7732   {
We use \str_if_eq:nnTF because it is fully expandable.
7733     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
7734     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
7735     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
7736     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
7737   }
7738 }

```

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`.

```

7739 \hook_gput_code:nnn { begindocument } { . }
7740 {
7741   \tl_set:Nn \l_@@_argspec_tl { m m m 0 { } E { _ ^ } { { } { } } }
7742   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
7743   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
7744     {
7745       \peek_remove_spaces:n
7746       {
7747         \@@_sub_matrix:nnnnnnn
7748         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
7749       }
7750     }
7751 }
```

The following macro will compute `\l_@@_first_i_tl`, `\l_@@_first_j_tl`, `\l_@@_last_i_tl` and `\l_@@_last_j_tl` from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

7752 \NewDocumentCommand \@@_compute_i_j:nn
7753 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7754 { \@@_compute_i_j:nnnn #1 #2 }

7755 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
7756 {
7757   \tl_set:Nn \l_@@_first_i_tl { #1 }
7758   \tl_set:Nn \l_@@_first_j_tl { #2 }
7759   \tl_set:Nn \l_@@_last_i_tl { #3 }
7760   \tl_set:Nn \l_@@_last_j_tl { #4 }
7761   \tl_if_eq:NnT \l_@@_first_i_tl { last }
7762     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
7763   \tl_if_eq:NnT \l_@@_first_j_tl { last }
7764     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
7765   \tl_if_eq:NnT \l_@@_last_i_tl { last }
7766     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
7767   \tl_if_eq:NnT \l_@@_last_j_tl { last }
7768     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
7769 }

7770 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
7771 {
7772   \group_begin:
```

The four following token lists correspond to the position of the `\SubMatrix`.

```

7773 \@@_compute_i_j:nn { #2 } { #3 }
7774 % added 6.19b
7775 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
7776 { \cs_set:Npn \arraystretch { 1 } }
7777 \bool_lazy_or:nnTF
7778 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7779 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7780 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
```

```

7781 {
7782   \str_clear_new:N \l_@@_submatrix_name_str
7783   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
7784   \pgfpicture
7785   \pgfrememberpicturepositiononpagetrue
7786   \pgf@relevantforpicturesizefalse
7787   \pgfset { inner~sep = \c_zero_dim }
7788   \dim_set_eq:Nn \l_@@_x_initial_dim \c_max_dim
7789   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

7790   \bool_if:NTF \l_@@_submatrix_slim_bool
7791   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
7792   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
7793   {
7794     \cs_if_exist:cT
7795     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7796     {
7797       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7798       \dim_set:Nn \l_@@_x_initial_dim
7799       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7800     }
7801     \cs_if_exist:cT
7802     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7803     {
7804       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7805       \dim_set:Nn \l_@@_x_final_dim
7806       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7807     }
7808   }
7809   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7810   { \@@_error:nn { Impossible~delimiter } { left } }
7811   {
7812     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7813     { \@@_error:nn { Impossible~delimiter } { right } }
7814     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7815   }
7816   \endpgfpicture
7817 }
7818 \group_end:
7819 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7820 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7821 {
7822   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7823   \dim_set:Nn \l_@@_y_initial_dim
7824   {
7825     \fp_to_dim:n
7826     {
7827       \pgf@y
7828       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
7829     }
7830   } % modified 6.13c
7831   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7832   \dim_set:Nn \l_@@_y_final_dim
7833   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
7834   % modified 6.13c
7835   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7836   {
7837     \cs_if_exist:cT
7838     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7839     {

```

```

7840         \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7841         \dim_set:Nn \l_@@_y_initial_dim
7842         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7843     }
7844     \cs_if_exist:cT
7845     { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7846     {
7847         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7848         \dim_set:Nn \l_@@_y_final_dim
7849         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7850     }
7851 }
7852 \dim_set:Nn \l_tmpa_dim
7853 {
7854     \l_@@_y_initial_dim - \l_@@_y_final_dim +
7855     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7856 }
7857 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7858     \group_begin:
7859     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7860     \@@_set_CT@arc@:V \l_@@_rules_color_tl
7861     \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`.

```

7862     \seq_map_inline:Nn \g_@@_cols_vlism_seq
7863     {
7864         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7865         {
7866             \int_compare:nNnT
7867             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
7868             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

7869             \@@_qpoint:n { col - ##1 }
7870             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7871             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7872             \pgfusepathqstroke
7873         }
7874     }
7875 }

```

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.

```

7876     \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7877     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7878     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7879     {
7880         \bool_lazy_and:nnTF
7881         { \int_compare_p:nNn { ##1 } > 0 }
7882         {
7883             \int_compare_p:nNn
7884             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7885         {
7886             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7887             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7888             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7889             \pgfusepathqstroke
7890         }
7891         { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
7892     }

```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:n` or `\clist_map_inline:Nn` is given by curryfication.

```

7893 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7894 { \int_step_inline:n { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7895 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7896 {
7897   \bool_lazy_and:nnTF
7898   { \int_compare_p:nNn { ##1 } > 0 }
7899   {
7900     \int_compare_p:nNn
7901     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7902   {
7903     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7904   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
7905   \dim_set:Nn \l_tmpa_dim
7906   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7907   \str_case:nn { #1 }
7908   {
7909     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7910     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7911     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7912     }
7913     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

7914   \dim_set:Nn \l_tmpb_dim
7915   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7916   \str_case:nn { #2 }
7917   {
7918     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7919     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7920     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7921   }
7922   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7923   \pgfusepathqstroke
7924   \group_end:
7925 }
7926 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7927 }

```

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).

```

7928 \str_if_empty:NF \l_@@_submatrix_name_str
7929 {
7930   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7931   \l_@@_x_initial_dim \l_@@_y_initial_dim
7932   \l_@@_x_final_dim \l_@@_y_final_dim
7933 }
7934 \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`.

```

7935 \begin { pgfscope }
7936 \pgftransformshift
7937 {
7938   \pgfpoint
7939   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7940   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7941 }

```

```

7942 \str_if_empty:NTF \l_@@_submatrix_name_str
7943 { \@@_node_left:nn #1 { } }
7944 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7945 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7946 \pgftransformshift
7947 {
7948   \pgfpoint
7949   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7950   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7951 }
7952 \str_if_empty:NTF \l_@@_submatrix_name_str
7953 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7954 {
7955   \@@_node_right:nnnn #2
7956   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7957 }
7958 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7959 \flag_clear_new:n { nicematrix }
7960 \l_@@_code_tl
7961 }

```

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`.

```

7962 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

7963 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
7964 {
7965   \use:e
7966   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
7967 }

```

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 -.

```

7968 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
7969 { #1 { \@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

7970 \tl_const:Nn \c_@@_integers_alist_tl
7971 {
7972   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7973   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7974   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7975   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7976 }
7977 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7978 {

```

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`.

```

7979 \tl_if_empty:nTF { #2 }
7980 {
7981   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7982   {
7983     \flag_raise:n { nicematrix }
7984     \int_if_even:nTF { \flag_height:n { nicematrix } }
7985     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7986     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7987   }
7988   { #1 }
7989 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

7990 { \c_@@_pgfpointanchor_iii:w { #1 } #2 }
7991 }

```

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. `\c_@@_pgfpointanchor_i:nn`).

```

7992 \cs_new:Npn \c_@@_pgfpointanchor_iii:w #1 #2 -
7993 {
7994   \str_case:nnF { #1 }
7995   {
7996     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7997     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7998   }

```

Now the case of a node of the form $i-j$.

```

7999 {
8000   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8001   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8002 }
8003 }

```

The command `\c_@@_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`).

```

8004 \cs_new_protected:Npn \c_@@_node_left:nn #1 #2
8005 {
8006   \pgfnode
8007   { rectangle }
8008   { east }
8009   {
8010     \nullfont
8011     \c_math_toggle_token
8012     \c_color:V \l_@@_delimiters_color_tl
8013     \left #1
8014     \vcenter
8015     {
8016       \nullfont
8017       \hrule \@height \l_tmpa_dim
8018       \c_depth \c_zero_dim
8019       \c_width \c_zero_dim
8020     }
8021     \right .
8022     \c_math_toggle_token
8023   }
8024   { #2 }

```

```

8025     { }
8026   }

```

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.

```

8027 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8028 {
8029   \pgfnode
8030     { rectangle }
8031     { west }
8032     {
8033       \nullfont
8034       \c_math_toggle_token
8035       \@@_color:V \l_@@_delimiters_color_tl
8036       \left .
8037       \vcenter
8038         {
8039           \nullfont
8040           \hrule \@height \l_tmpa_dim
8041             \@depth \c_zero_dim
8042             \@width \c_zero_dim
8043         }
8044       \right #1
8045       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8046       ^ { \smash { #4 } }
8047       \c_math_toggle_token
8048     }
8049     { #2 }
8050     { }
8051 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8052 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8053 {
8054   \peek_remove_spaces:n
8055   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8056 }
8057 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8058 {
8059   \peek_remove_spaces:n
8060   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8061 }
8062 \keys_define:nn { NiceMatrix / Brace }
8063 {
8064   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8065   left-shorten .default:n = true ,
8066   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8067   shorten .meta:n = { left-shorten , right-shorten } ,
8068   right-shorten .default:n = true ,
8069   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8070   yshift .value_required:n = true ,
8071   yshift .initial:n = \c_zero_dim ,
8072   color .tl_set:N = \l_tmpa_tl ,
8073   color .value_required:n = true ,

```

```

8074     unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
8075 }

```

#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`.

```

8076 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8077 {
8078   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8079   \@@_compute_i_j:nn { #1 } { #2 }
8080   \bool_lazy_or:nnTF
8081     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8082     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8083     {
8084       \str_if_eq:nnTF { #5 } { under }
8085       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8086       { \@@_error:nn { Construct-too-large } { \OverBrace } }
8087     }
8088     {
8089       \tl_clear:N \l_tmpa_tl
8090       \keys_set:nn { NiceMatrix / Brace } { #4 }
8091       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8092       \pgfpicture
8093       \pgfrememberpicturepositiononpagetrue
8094       \pgf@relevantforpicturesizefalse
8095       \bool_if:NT \l_@@_brace_left_shorten_bool
8096       {
8097         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8098         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8099         {
8100           \cs_if_exist:cT
8101             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8102             {
8103               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8104               \dim_set:Nn \l_@@_x_initial_dim
8105                 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8106             }
8107         }
8108       }
8109       \bool_lazy_or:nnT
8110         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8111         { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8112         {
8113           \@@_qpoint:n { col - \l_@@_first_j_tl }
8114           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8115         }
8116       \bool_if:NT \l_@@_brace_right_shorten_bool
8117       {
8118         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8119         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8120         {
8121           \cs_if_exist:cT
8122             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8123             {
8124               \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8125               \dim_set:Nn \l_@@_x_final_dim
8126                 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8127             }
8128         }
8129       }
8130       \bool_lazy_or:nnT
8131         { \bool_not_p:n \l_@@_brace_right_shorten_bool }

```



```

8132     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8133     {
8134         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8135         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8136     }
8137     \pgfset { inner~sep = \c_zero_dim }
8138     \str_if_eq:nnTF { #5 } { under }
8139     { \@@_underbrace_i:n { #3 } }
8140     { \@@_overbrace_i:n { #3 } }
8141     \endpgfpicture
8142 }
8143 \group_end:
8144 }

```

The argument is the text to put above the brace.

```

8145 \cs_new_protected:Npn \@@_overbrace_i:n #1
8146 {
8147     \@@_qpoint:n { row - \l_@@_first_i_tl }
8148     \pgftransformshift
8149     {
8150         \pgfpoint
8151         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8152         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8153     }
8154     \pgfnode
8155     { rectangle }
8156     { south }
8157     {
8158         \vbox_top:n
8159         {
8160             \group_begin:
8161             \everycr { }
8162             \halign
8163             {
8164                 \hfil ## \hfil \crrc
8165                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8166                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8167                 \c_math_toggle_token
8168                 \overbrace
8169                 {
8170                     \hbox_to_wd:nn
8171                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8172                     { }
8173                 }
8174                 \c_math_toggle_token
8175                 \cr
8176             }
8177             \group_end:
8178         }
8179     }
8180     { }
8181     { }
8182 }

```

The argument is the text to put under the brace.

```

8183 \cs_new_protected:Npn \@@_underbrace_i:n #1
8184 {
8185     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8186     \pgftransformshift
8187     {
8188         \pgfpoint
8189         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8190         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }

```

```

8191     }
8192     \pgfnode
8193     { rectangle }
8194     { north }
8195     {
8196         \group_begin:
8197         \everycr { }
8198         \vbox:n
8199         {
8200             \halign
8201             {
8202                 \hfil ## \hfil \crrc
8203                 \c_math_toggle_token
8204                 \underbrace
8205                 {
8206                     \hbox_to_wd:nn
8207                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8208                     { }
8209                 }
8210                 \c_math_toggle_token
8211                 \cr
8212                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8213                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8214             }
8215         }
8216     \group_end:
8217     }
8218     { }
8219     { }
8220 }

```

35 The command \ShowCellNames

```

8221 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8222 {
8223     \dim_zero_new:N \g_@@_tmpc_dim
8224     \dim_zero_new:N \g_@@_tmpd_dim
8225     \dim_zero_new:N \g_@@_tmpe_dim
8226     \int_step_inline:nn \c@iRow
8227     {
8228         \begin { pgfpicture }
8229         \@@_qpoint:n { row - ##1 }
8230         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8231         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8232         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8233         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8234         \bool_if:NTF \l_@@_in_code_after_bool
8235         \end { pgfpicture }
8236         \int_step_inline:nn \c@jCol
8237         {
8238             \hbox_set:Nn \l_tmpa_box
8239             { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
8240             \begin { pgfpicture }
8241             \@@_qpoint:n { col - ####1 }
8242             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8243             \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8244             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8245             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8246             \endpgfpicture

```

```

8247 \end { pgfpicture }
8248 \fp_set:Nn \l_tmpa_fp
8249 {
8250 \fp_min:nn
8251 {
8252 \fp_min:nn
8253 { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8254 { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8255 }
8256 { 1.0 }
8257 }
8258 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8259 \pgfpicture
8260 \pgfrememberpicturepositiononpagetrue
8261 \pgf@relevantforpicturesizefalse
8262 \pgftransformshift
8263 {
8264 \pgfpoint
8265 { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8266 { \dim_use:N \g_tmpa_dim }
8267 }
8268 \pgfnode
8269 { rectangle }
8270 { center }
8271 { \box_use:N \l_tmpa_box }
8272 { }
8273 { }
8274 \endpgfpicture
8275 }
8276 }
8277 }
8278 \NewDocumentCommand \@@_ShowCellNames { }
8279 {
8280 \bool_if:NT \l_@@_in_code_after_bool
8281 {
8282 \pgfpicture
8283 \pgfrememberpicturepositiononpagetrue
8284 \pgf@relevantforpicturesizefalse
8285 \pgfpathrectanglecorners
8286 { \@@_qpoint:n { 1 } }
8287 {
8288 \@@_qpoint:n
8289 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8290 }
8291 \pgfsetfillopacity { 0.75 }
8292 \pgfsetfillcolor { white }
8293 \pgfusepathqfill
8294 \endpgfpicture
8295 }
8296 \dim_zero_new:N \g_@@_tmpc_dim
8297 \dim_zero_new:N \g_@@_tmpd_dim
8298 \dim_zero_new:N \g_@@_tmpe_dim
8299 \int_step_inline:nn \c@iRow
8300 {
8301 \bool_if:NTF \l_@@_in_code_after_bool
8302 {
8303 \pgfpicture
8304 \pgfrememberpicturepositiononpagetrue
8305 \pgf@relevantforpicturesizefalse
8306 }
8307 { \begin { pgfpicture } }
8308 \@@_qpoint:n { row - ##1 }
8309 \dim_set_eq:NN \l_tmpa_dim \pgf@y

```

```

8310 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8311 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8312 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8313 \bool_if:NTF \l_@@_in_code_after_bool
8314 { \endpgfpicture }
8315 { \end { pgfpicture } }
8316 \int_step_inline:nn \c@jCol
8317 {
8318   \hbox_set:Nn \l_tmpa_box
8319   {
8320     \normalfont \Large \sffamily \bfseries
8321     \bool_if:NTF \l_@@_in_code_after_bool
8322     { \color { red } }
8323     { \color { red ! 50 } }
8324     ##1 - ####1
8325   }
8326   \bool_if:NTF \l_@@_in_code_after_bool
8327   {
8328     \pgfpicture
8329     \pgfrememberpicturepositiononpagetrue
8330     \pgf@relevantforpicturesizefalse
8331   }
8332   { \begin { pgfpicture } }
8333   \@@_qpoint:n { col - ####1 }
8334   \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8335   \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8336   \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8337   \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8338   \bool_if:NTF \l_@@_in_code_after_bool
8339   { \endpgfpicture }
8340   { \end { pgfpicture } }
8341   \fp_set:Nn \l_tmpa_fp
8342   {
8343     \fp_min:nn
8344     {
8345       \fp_min:nn
8346       { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8347       { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8348     }
8349     { 1.0 }
8350   }
8351   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8352   \pgfpicture
8353   \pgfrememberpicturepositiononpagetrue
8354   \pgf@relevantforpicturesizefalse
8355   \pgftransformshift
8356   {
8357     \pgfpoint
8358     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8359     { \dim_use:N \g_tmpa_dim }
8360   }
8361   \pgfnode
8362   { rectangle }
8363   { center }
8364   { \box_use:N \l_tmpa_box }
8365   { }
8366   { }
8367   \endpgfpicture
8368 }
8369 }
8370 }

```

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.

```
8371 \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.

```
8372 \bool_new:N \c_@@_footnote_bool
8373 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
8374 {
8375   The-key~'\l_keys_key_str'~is-unknown. \\
8376   That-key~will-be-ignored. \\
8377   For-a-list-of-the-available-keys,~type-H~<return>.
8378 }
8379 {
8380   The-available-keys-are~(in-alphabetic-order):~
8381   footnote,~
8382   footnotehyper,~
8383   messages-for-Overleaf,~
8384   no-test-for-array,~
8385   renew-dots,~and
8386   renew-matrix.
8387 }
8388 \keys_define:nn { NiceMatrix / Package }
8389 {
8390   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8391   renew-dots .value_forbidden:n = true ,
8392   renew-matrix .code:n = \@@_renew_matrix: ,
8393   renew-matrix .value_forbidden:n = true ,
8394   messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
8395   footnote .bool_set:N = \c_@@_footnote_bool ,
8396   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
8397   no-test-for-array .bool_set:N = \c_@@_no_test_for_array_bool ,
8398   no-test-for-array .default:n = true ,
8399   unknown .code:n = \@@_error:n { Unknown-key-for-package }
8400 }
8401 \ProcessKeysOptions { NiceMatrix / Package }

8402 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8403 {
8404   You-can't-use-the-option-'footnote'~because-the-package~
8405   footnotehyper~has~already~been~loaded.~
8406   If-you-want,~you-can-use-the-option-'footnotehyper'~and-the-footnotes~
8407   within-the-environments-of-nicematrix~will-be-extracted-with-the-tools~
8408   of-the-package-footnotehyper.\\
8409   The~package~footnote~won't-be-loaded.
8410 }
8411 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8412 {
8413   You-can't-use-the-option-'footnotehyper'~because-the-package~
8414   footnote~has~already~been~loaded.~
8415   If-you-want,~you-can-use-the-option-'footnote'~and-the-footnotes~
8416   within-the-environments-of-nicematrix~will-be-extracted-with-the-tools~
8417   of-the-package-footnote.\\
```

```

8418   The~package~footnotehyper~won't~be~loaded.
8419 }

```

```

8420 \bool_if:NT \c_@@_footnote_bool
8421 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8422   \IfClassLoadedTF { beamer }
8423   { \bool_set_false:N \c_@@_footnote_bool }
8424   {
8425     \IfPackageLoadedTF { footnotehyper }
8426     { \@@_error:n { footnote~with~footnotehyper~package } }
8427     { \usepackage { footnote } }
8428   }
8429 }

8430 \bool_if:NT \c_@@_footnotehyper_bool
8431 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8432   \IfClassLoadedTF { beamer }
8433   { \bool_set_false:N \c_@@_footnote_bool }
8434   {
8435     \IfPackageLoadedTF { footnote }
8436     { \@@_error:n { footnotehyper~with~footnote~package } }
8437     { \usepackage { footnotehyper } }
8438   }
8439   \bool_set_true:N \c_@@_footnote_bool
8440 }

```

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.

```

8441 \bool_new:N \l_@@_underscore_loaded_bool
8442 \IfPackageLoadedTF { underscore }
8443 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8444 { }

8445 \hook_gput_code:nnn { begindocument } { . }
8446 {
8447   \bool_if:NF \l_@@_underscore_loaded_bool
8448   {
8449     \IfPackageLoadedTF { underscore }
8450     { \@@_error:n { underscore~after~nicematrix } }
8451     { }
8452   }
8453 }

```

38 Error messages of the package

```

8454 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
8455 { \str_const:Nn \c_@@_available_keys_str { } }
8456 {
8457   \str_const:Nn \c_@@_available_keys_str
8458   { For~a~list~of~the~available~keys,~type~H~<return>. }
8459 }

8460 \seq_new:N \g_@@_types_of_matrix_seq
8461 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8462 {
8463   NiceMatrix ,
8464   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8465 }
8466 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8467 { \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`.

```

8468 \cs_new_protected:Npn \@@_error_too_much_cols:
8469 {
8470   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8471   {
8472     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8473     { \@@_fatal:n { too-much-cols-for-matrix } }
8474     {
8475       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8476       { \@@_fatal:n { too-much-cols-for-matrix } }
8477       {
8478         \bool_if:NF \l_@@_last_col_without_value_bool
8479         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
8480       }
8481     }
8482   }
8483   {
8484     \IfPackageLoadedTF { tabularx }
8485     {
8486       \str_if_eq:VnTF \g_@@_name_env_str { NiceTabularX }
8487       {
8488         \int_compare:nNnTF \c@iRow = \c_zero_int
8489         { \@@_fatal:n { X-columns-with-tabularx } }
8490         {
8491           \@@_fatal:nn { too-much-cols-for-array }
8492           {
8493             However,~this-message-may-be-erroneous:~
8494             maybe-you-have-used-X-columns-while-'tabularx'~is~loaded,~
8495             ~which-is-forbidden~(however,~it's~still~possible~to~use~
8496             X-columns-in~{NiceTabularX}).
8497           }
8498         }
8499       }
8500       { \@@_fatal:nn { too-much-cols-for-array } { } }
8501     }
8502     { \@@_fatal:nn { too-much-cols-for-array } { } }
8503   }
8504 }

```

The following command must *not* be protected since it's used in an error message.

```

8505 \cs_new:Npn \@@_message_hdotsfor:
8506 {

```

```

8507 \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8508 { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is~incorrect.}
8509 }
8510 \@@_msg_new:nn { negative-weight }
8511 {
8512   Negative-weight.\
8513   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
8514   the~value~'\int_use:N \l_@@_weight_int'.\
8515   The~absolute~value~will~be~used.
8516 }
8517 \@@_msg_new:nn { last-col-not-used }
8518 {
8519   Column~not~used.\
8520   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
8521   in~your~\@@_full_name_env:~.~However,~you~can~go~on.
8522 }
8523 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
8524 {
8525   Too~much~columns.\
8526   In~the~row~\int_eval:n { \c@iRow },~
8527   you~try~to~use~more~columns~
8528   than~allowed~by~your~\@@_full_name_env:~.\@@_message_hdotsfor:\
8529   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
8530   (plus~the~exterior~columns).~This~error~is~fatal.
8531 }
8532 \@@_msg_new:nn { too-much-cols-for-matrix }
8533 {
8534   Too~much~columns.\
8535   In~the~row~\int_eval:n { \c@iRow },~
8536   you~try~to~use~more~columns~than~allowed~by~your~
8537   \@@_full_name_env:~.\@@_message_hdotsfor:\ Recall~that~the~maximal~
8538   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
8539   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
8540   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
8541   \token_to_str:N \setcounter\ to~change~that~value).~
8542   This~error~is~fatal.
8543 }
8544 \@@_msg_new:nn { too-much-cols-for-array }
8545 {
8546   Too~much~columns.\
8547   In~the~row~\int_eval:n { \c@iRow },~
8548   ~you~try~to~use~more~columns~than~allowed~by~your~
8549   \@@_full_name_env:~.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
8550   \int_use:N \g_@@_static_num_of_col_int\
8551   ~(plus~the~potential~exterior~ones).~#1
8552   This~error~is~fatal.
8553 }
8554 \@@_msg_new:nn { X-columns-with-tabularx }
8555 {
8556   There~is~a~problem.\
8557   You~have~probably~used~X~columns~in~your~environment~{\g_@@_name_env_str}.~
8558   That's~not~allowed~because~'tabularx'~is~loaded~(however,~you~can~use~X~columns~
8559   in~an~environment~{NiceTabularX}).\
8560   This~error~is~fatal.
8561 }
8562 \@@_msg_new:nn { columns-not-used }
8563 {
8564   Columns~not~used.\
8565   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8566   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\

```



```

8567 The~columns~you~did~not~used~won't~be~created.\\
8568 We~won't~have~similar~error~till~the~end~of~the~document.
8569 }

8570 \@@_msg_new:nn { in~first~col }
8571 {
8572   Erroneous~use.\\
8573   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
8574   That~command~will~be~ignored.
8575 }

8576 \@@_msg_new:nn { in~last~col }
8577 {
8578   Erroneous~use.\\
8579   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
8580   That~command~will~be~ignored.
8581 }

8582 \@@_msg_new:nn { in~first~row }
8583 {
8584   Erroneous~use.\\
8585   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
8586   That~command~will~be~ignored.
8587 }

8588 \@@_msg_new:nn { in~last~row }
8589 {
8590   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
8591   That~command~will~be~ignored.
8592 }

8593 \@@_msg_new:nn { caption~outside~float }
8594 {
8595   Key~caption~forbidden.\\
8596   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
8597   environment.~This~key~will~be~ignored.
8598 }

8599 \@@_msg_new:nn { short~caption~without~caption }
8600 {
8601   You~should~not~use~the~key~'short~caption'~without~'caption'.~
8602   However,~your~'short~caption'~will~be~used~as~'caption'.
8603 }

8604 \@@_msg_new:nn { double~closing~delimiter }
8605 {
8606   Double~delimiter.\\
8607   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
8608   delimiter.~This~delimiter~will~be~ignored.
8609 }

8610 \@@_msg_new:nn { delimiter~after~opening }
8611 {
8612   Double~delimiter.\\
8613   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
8614   delimiter.~That~delimiter~will~be~ignored.
8615 }

8616 \@@_msg_new:nn { bad~option~for~line~style }
8617 {
8618   Bad~line~style.\\
8619   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
8620   is~'standard'.~That~key~will~be~ignored.
8621 }

8622 \@@_msg_new:nn { Identical~notes~in~caption }
8623 {
8624   Identical~tabular~notes.\\
8625   You~can't~put~several~notes~with~the~same~content~in~

```

```

8626 \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\
8627 If~you~go~on,~the~output~will~probably~be~erroneous.
8628 }

8629 \@@_msg_new:nn { tabularnote~below~the~tabular }
8630 {
8631 \token_to_str:N \tabularnote\ forbidden\
8632 You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8633 of~your~tabular~because~the~caption~will~be~composed~below~
8634 the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8635 key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\
8636 Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8637 no~similar~error~will~raised~in~this~document.
8638 }

8639 \@@_msg_new:nn { Unknown~key~for~rules }
8640 {
8641 Unknown~key.\
8642 There~is~only~two~keys~available~here:~width~and~color.\
8643 You~key~'\l_keys_key_str'~will~be~ignored.
8644 }

8645 \@@_msg_new:nnn { Unknown~key~for~custom~line }
8646 {
8647 Unknown~key.\
8648 The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
8649 It~you~go~on,~you~will~probably~have~other~errors. \
8650 \c_@@_available_keys_str
8651 }
8652 {
8653 The~available~keys~are~(in~alphabetic~order):~
8654 ccommand,~
8655 color,~
8656 command,~
8657 dotted,~
8658 letter,~
8659 multiplicity,~
8660 sep-color,~
8661 tikz,~and~total~width.
8662 }

8663 \@@_msg_new:nnn { Unknown~key~for~xdots }
8664 {
8665 Unknown~key.\
8666 The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\
8667 \c_@@_available_keys_str
8668 }
8669 {
8670 The~available~keys~are~(in~alphabetic~order):~
8671 'color',~
8672 'horizontal~labels',~
8673 'inter',~
8674 'line~style',~
8675 'radius',~
8676 'shorten',~
8677 'shorten~end'~and~'shorten~start'.
8678 }

8679 \@@_msg_new:nn { Unknown~key~for~rowcolors }
8680 {
8681 Unknown~key.\
8682 As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
8683 (and~you~try~to~use~'\l_keys_key_str')\
8684 That~key~will~be~ignored.
8685 }

8686 \@@_msg_new:nn { label~without~caption }

```

```

8687 {
8688   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
8689   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
8690 }
8691 \@@_msg_new:nn { W~warning }
8692 {
8693   Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
8694   (row~\int_use:N \c@iRow).
8695 }
8696 \@@_msg_new:nn { Construct~too~large }
8697 {
8698   Construct~too~large.\\
8699   Your~command~\token_to_str:N #1
8700   can't~be~drawn~because~your~matrix~is~too~small.\\
8701   That~command~will~be~ignored.
8702 }
8703 \@@_msg_new:nn { underscore~after~nicematrix }
8704 {
8705   Problem~with~'underscore'.\\
8706   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
8707   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
8708   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
8709 }
8710 \@@_msg_new:nn { ampersand~in~light~syntax }
8711 {
8712   Ampersand~forbidden.\\
8713   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
8714   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
8715 }
8716 \@@_msg_new:nn { double~backslash~in~light~syntax }
8717 {
8718   Double~backslash~forbidden.\\
8719   You~can't~use~\token_to_str:N
8720   \\~to~separate~rows~because~the~key~'light-syntax'~
8721   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
8722   (set~by~the~key~'end-of-row').~This~error~is~fatal.
8723 }
8724 \@@_msg_new:nn { hlines~with~color }
8725 {
8726   Incompatible~keys.\\
8727   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
8728   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
8729   Maybe~it~will~possible~in~future~version.\\
8730   Your~key~will~be~discarded.
8731 }
8732 \@@_msg_new:nn { bad~value~for~baseline }
8733 {
8734   Bad~value~for~baseline.\\
8735   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
8736   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
8737   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
8738   the~form~'line-i'.\\
8739   A~value~of~1~will~be~used.
8740 }
8741 \@@_msg_new:nn { ragged2e~not~loaded }
8742 {
8743   You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
8744   your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
8745   \l_keys_key_str'~will~be~used~instead.
8746 }

```

```

8747 \@@_msg_new:nn { Invalid-name }
8748 {
8749   Invalid-name.\
8750   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
8751   \SubMatrix\ of~your~\@@_full_name_env:.\
8752   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
8753   This~key~will~be~ignored.
8754 }

8755 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
8756 {
8757   Wrong-line.\
8758   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
8759   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
8760   number~is~not~valid.~It~will~be~ignored.
8761 }

8762 \@@_msg_new:nn { Impossible-delimiter }
8763 {
8764   Impossible-delimiter.\
8765   It's~impossible~to~draw~the~#1~delimiter~of~your~
8766   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
8767   in~that~column.
8768   \bool_if:NT \l_@@_submatrix_slim_bool
8769   { ~Maybe~you~should~try~without~the~key~'slim'. } \
8770   This~\token_to_str:N \SubMatrix\ will~be~ignored.
8771 }

8772 \@@_msg_new:nn { width-without-X-columns }
8773 {
8774   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
8775   That~key~will~be~ignored.
8776 }

8777 \@@_msg_new:nn { key-multiplicity-with-dotted }
8778 {
8779   Incompatible~keys. \
8780   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
8781   in~a~'custom-line'.~They~are~incompatible. \
8782   The~key~'multiplicity'~will~be~discarded.
8783 }

8784 \@@_msg_new:nn { empty-environment }
8785 {
8786   Empty-environment.\
8787   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
8788 }

8789 \@@_msg_new:nn { No-letter~and~no-command }
8790 {
8791   Erroneous-use.\
8792   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
8793   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
8794   '~'ccommand'~(to~draw~horizontal~rules).\
8795   However,~you~can~go~on.
8796 }

8797 \@@_msg_new:nn { Forbidden-letter }
8798 {
8799   Forbidden-letter.\
8800   You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\
8801   It~will~be~ignored.
8802 }

8803 \@@_msg_new:nn { Several-letters }
8804 {
8805   Wrong-name.\
8806   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~

```

```

8807     have-used~'\l_@@_letter_str').\\
8808     It-will-be-ignored.
8809 }

8810 \@@_msg_new:nn { Delimiter-with-small }
8811 {
8812     Delimiter-forbidden.\\
8813     You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\
8814     because-the-key~'small'~is-in-force.\\
8815     This-error-is-fatal.
8816 }

8817 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
8818 {
8819     Unknown-cell.\\
8820     Your-command~\token_to_str:N\line\{#1\}\{#2\}~in-
8821     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
8822     can't-be-executed-because-a-cell-doesn't-exist.\\
8823     This-command~\token_to_str:N \line\ will-be-ignored.
8824 }

8825 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
8826 {
8827     Duplicate-name.\\
8828     The~name~'#1'~is-already-used-for-a~\token_to_str:N \SubMatrix\
8829     in~this~\@@_full_name_env:.\
8830     This-key-will-be-ignored.\\
8831     \bool_if:NF \c_@@_messages_for_Overleaf_bool
8832     { For-a-list-of-the-names-already-used,~type-H-<return>. }
8833 }
8834 {
8835     The-names-already-defined-in-this~\@@_full_name_env:\ are:~
8836     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
8837 }

8838 \@@_msg_new:nn { r-or-l-with-preamble }
8839 {
8840     Erroneous-use.\\
8841     You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~
8842     You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
8843     your~\@@_full_name_env:.\
8844     This-key-will-be-ignored.
8845 }

8846 \@@_msg_new:nn { Hdotsfor-in-col-0 }
8847 {
8848     Erroneous-use.\\
8849     You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior-column-of~
8850     the-array.~This-error-is-fatal.
8851 }

8852 \@@_msg_new:nn { bad-corner }
8853 {
8854     Bad-corner.\\
8855     #1-is-an-incorrect-specification-for-a-corner~(in-the-key~
8856     'corners').~The-available-values-are:~NW,~SW,~NE-and-SE.\\
8857     This-specification-of-corner-will-be-ignored.
8858 }

8859 \@@_msg_new:nn { bad-border }
8860 {
8861     Bad-border.\\
8862     \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
8863     (in-the-key~'borders'~of-the-command~\token_to_str:N \Block).~
8864     The-available-values-are:~left,~right,~top-and-bottom~(and-you-can~
8865     also-use-the-key~'tikz'
8866     \IfPackageLoadedTF { tikz }
8867     { }

```

```

8868     {\if~you~load~the~LaTeX~package~'tikz'}).\}
8869     This~specification~of~border~will~be~ignored.
8870 }

8871 \@@_msg_new:nn { tikz~key~without~tikz }
8872 {
8873     Tikz~not~loaded.\}
8874     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
8875     \Block'~because~you~have~not~loaded~tikz.~
8876     This~key~will~be~ignored.
8877 }

8878 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
8879 {
8880     Erroneous~use.\}
8881     In~the~\@@_full_name_env:,~you~must~use~the~key~
8882     'last-col'~without~value.\}
8883     However,~you~can~go~on~for~this~time~
8884     (the~value~'\l_keys_value_tl'~will~be~ignored).
8885 }

8886 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
8887 {
8888     Erroneous~use.\}
8889     In~\NiceMatrixoptions,~you~must~use~the~key~
8890     'last-col'~without~value.\}
8891     However,~you~can~go~on~for~this~time~
8892     (the~value~'\l_keys_value_tl'~will~be~ignored).
8893 }

8894 \@@_msg_new:nn { Block~too~large~1 }
8895 {
8896     Block~too~large.\}
8897     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
8898     too~small~for~that~block. \}
8899 }

8900 \@@_msg_new:nn { Block~too~large~2 }
8901 {
8902     Block~too~large.\}
8903     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8904     \g_@@_static_num_of_col_int\
8905     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
8906     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
8907     (&)~at~the~end~of~the~first~row~of~your~
8908     \@@_full_name_env:.\}
8909     This~block~and~maybe~others~will~be~ignored.
8910 }

8911 \@@_msg_new:nn { unknown~column~type }
8912 {
8913     Bad~column~type.\}
8914     The~column~type~'#1'~in~your~\@@_full_name_env:\
8915     is~unknown. \}
8916     This~error~is~fatal.
8917 }

8918 \@@_msg_new:nn { unknown~column~type~S }
8919 {
8920     Bad~column~type.\}
8921     The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \}
8922     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
8923     load~that~package. \}
8924     This~error~is~fatal.
8925 }

8926 \@@_msg_new:nn { tabularnote~forbidden }
8927 {

```

```

8928 Forbidden~command.\\
8929 You~can't~use~the~command~\token_to_str:N\tabularnote\
8930 ~here.~This~command~is~available~only~in~
8931 \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
8932 the~argument~of~a~command~\token_to_str:N \caption\ included~
8933 in~an~environment~{table}. \\
8934 This~command~will~be~ignored.
8935 }
8936 \@@_msg_new:nn { borders~forbidden }
8937 {
8938   Forbidden~key.\\
8939   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
8940   because~the~option~'rounded-corners'~
8941   is~in~force~with~a~non-zero~value.\\
8942   This~key~will~be~ignored.
8943 }
8944 \@@_msg_new:nn { bottomrule~without~booktabs }
8945 {
8946   booktabs~not~loaded.\\
8947   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
8948   loaded~'booktabs'.\\
8949   This~key~will~be~ignored.
8950 }
8951 \@@_msg_new:nn { enumitem~not~loaded }
8952 {
8953   enumitem~not~loaded.\\
8954   You~can't~use~the~command~\token_to_str:N\tabularnote\
8955   ~because~you~haven't~loaded~'enumitem'.\\
8956   All~the~commands~\token_to_str:N\tabularnote\ will~be~
8957   ignored~in~the~document.
8958 }
8959 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
8960 {
8961   Tikz~not~loaded.\\
8962   You~have~used~the~key~'tikz'~in~the~definition~of~a~
8963   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
8964   You~can~go~on~but~you~will~have~another~error~if~you~actually~
8965   use~that~custom~line.
8966 }
8967 \@@_msg_new:nn { tikz~in~borders~without~tikz }
8968 {
8969   Tikz~not~loaded.\\
8970   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
8971   command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
8972   That~key~will~be~ignored.
8973 }
8974 \@@_msg_new:nn { color~in~custom~line~with~tikz }
8975 {
8976   Erroneous~use.\\
8977   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
8978   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
8979   The~key~'color'~will~be~discarded.
8980 }
8981 \@@_msg_new:nn { Wrong~last~row }
8982 {
8983   Wrong~number.\\
8984   You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
8985   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
8986   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
8987   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
8988   without~value~(more~compilations~might~be~necessary).

```

```

8989 }
8990 \@@_msg_new:nn { Yet~in~env }
8991 {
8992   Nested~environments.\\
8993   Environments~of~nicematrix~can't~be~nested.\\
8994   This~error~is~fatal.
8995 }
8996 \@@_msg_new:nn { Outside~math~mode }
8997 {
8998   Outside~math~mode.\\
8999   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9000   (and~not~in~\token_to_str:N \vcenter).\\
9001   This~error~is~fatal.
9002 }
9003 \@@_msg_new:nn { One~letter~allowed }
9004 {
9005   Bad~name.\\
9006   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9007   It~will~be~ignored.
9008 }
9009 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9010 {
9011   Environment~{TabularNote}~forbidden.\\
9012   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9013   but~*before*~the~\token_to_str:N \CodeAfter.\\
9014   This~environment~{TabularNote}~will~be~ignored.
9015 }
9016 \@@_msg_new:nn { varwidth~not~loaded }
9017 {
9018   varwidth~not~loaded.\\
9019   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9020   loaded.\\
9021   Your~column~will~behave~like~'p'.
9022 }
9023 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9024 {
9025   Unknow~key.\\
9026   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9027   \c_@@_available_keys_str
9028 }
9029 {
9030   The~available~keys~are~(in~alphabetic~order):~
9031   color,~
9032   dotted,~
9033   multiplicity,~
9034   sep-color,~
9035   tikz,~and~total-width.
9036 }
9037
9038 \@@_msg_new:nnn { Unknown~key~for~Block }
9039 {
9040   Unknown~key.\\
9041   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9042   \Block.\\ It~will~be~ignored. \\
9043   \c_@@_available_keys_str
9044 }
9045 {
9046   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9047   hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,~
9048   t,~T,~tikz,~transparent~and~vlines.
9049 }

```



```

9050 \@@_msg_new:nn { Version-of-siunitx-too-old }
9051 {
9052   siunitx-too-old.\
9053   You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9054   is~too~old.~You~need~at~least~v~3.0.38~and~your~log~file~says:~"siunitx,~
9055   \use:c { ver @ siunitx.sty }". \
9056   This~error~is~fatal.
9057 }
9058 \@@_msg_new:nnn { Unknown~key~for~Brace }
9059 {
9060   Unknown~key.\
9061   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9062   \UnderBrace\ and~\token_to_str:N \OverBrace.\
9063   It~will~be~ignored. \
9064   \c_@@_available_keys_str
9065 }
9066 {
9067   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9068   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9069   right~shorten)~and~yshift.
9070 }
9071 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9072 {
9073   Unknown~key.\
9074   The~key~'\l_keys_key_str'~is~unknown.\
9075   It~will~be~ignored. \
9076   \c_@@_available_keys_str
9077 }
9078 {
9079   The~available~keys~are~(in~alphabetic~order):~
9080   delimiters/color,~
9081   rules~(with~the~subkeys~'color'~and~'width'),~
9082   sub-matrix~(several~subkeys)~
9083   and~xdots~(several~subkeys).~
9084   The~latter~is~for~the~command~\token_to_str:N \line.
9085 }
9086 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9087 {
9088   Unknown~key.\
9089   The~key~'\l_keys_key_str'~is~unknown.\
9090   It~will~be~ignored. \
9091   \c_@@_available_keys_str
9092 }
9093 {
9094   The~available~keys~are~(in~alphabetic~order):~
9095   create-cell-nodes,~
9096   delimiters/color~and~
9097   sub-matrix~(several~subkeys).
9098 }
9099 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9100 {
9101   Unknown~key.\
9102   The~key~'\l_keys_key_str'~is~unknown.\
9103   That~key~will~be~ignored. \
9104   \c_@@_available_keys_str
9105 }
9106 {
9107   The~available~keys~are~(in~alphabetic~order):~
9108   'delimiters/color',~
9109   'extra-height',~
9110   'hlines',~
9111   'hvlines',~

```

```

9112     'left-xshift',~
9113     'name',~
9114     'right-xshift',~
9115     'rules'~(with~the~subkeys~'color'~and~'width'),~
9116     'slim',~
9117     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9118     and~'right-xshift').\\
9119 }
9120 \@@_msg_new:nnn { Unknown~key~for~notes }
9121 {
9122     Unknown~key.\\
9123     The~key~'\l_keys_key_str'~is~unknown.\\
9124     That~key~will~be~ignored. \\
9125     \c_@@_available_keys_str
9126 }
9127 {
9128     The~available~keys~are~(in~alphabetic~order):~
9129     bottomrule,~
9130     code~after,~
9131     code~before,~
9132     detect~duplicates,~
9133     enumitem~keys,~
9134     enumitem~keys~para,~
9135     para,~
9136     label~in~list,~
9137     label~in~tabular~and~
9138     style.
9139 }
9140 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9141 {
9142     Unknown~key.\\
9143     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9144     \token_to_str:N \RowStyle. \\
9145     That~key~will~be~ignored. \\
9146     \c_@@_available_keys_str
9147 }
9148 {
9149     The~available~keys~are~(in~alphabetic~order):~
9150     'bold',~
9151     'cell-space-top-limit',~
9152     'cell-space-bottom-limit',~
9153     'cell-space-limits',~
9154     'color',~
9155     'nb-rows'~and~
9156     'rowcolor'.
9157 }
9158 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9159 {
9160     Unknown~key.\\
9161     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9162     \token_to_str:N \NiceMatrixOptions. \\
9163     That~key~will~be~ignored. \\
9164     \c_@@_available_keys_str
9165 }
9166 {
9167     The~available~keys~are~(in~alphabetic~order):~
9168     allow~duplicate~names,~
9169     caption~above,~
9170     cell-space-bottom-limit,~
9171     cell-space-limits,~
9172     cell-space-top-limit,~
9173     code~for~first~col,~
9174     code~for~first~row,~

```

```

9175 code-for-last-col,~
9176 code-for-last-row,~
9177 corners,~
9178 custom-key,~
9179 create-extra-nodes,~
9180 create-medium-nodes,~
9181 create-large-nodes,~
9182 delimiters~(several~subkeys),~
9183 end-of-row,~
9184 first-col,~
9185 first-row,~
9186 hlines,~
9187 hvlines,~
9188 hvlines-except-borders,~
9189 last-col,~
9190 last-row,~
9191 left-margin,~
9192 light-syntax,~
9193 matrix/columns-type,~
9194 notes~(several~subkeys),~
9195 nullify-dots,~
9196 pgf-node-code,~
9197 renew-dots,~
9198 renew-matrix,~
9199 respect-arraystretch,~
9200 right-margin,~
9201 rules~(with~the~subkeys~'color'~and~'width'),~
9202 small,~
9203 sub-matrix~(several~subkeys),~
9204 vlines,~
9205 xdots~(several~subkeys).
9206 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

9207 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9208 {
9209   Unknown~key.\\
9210   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9211   \{NiceArray\}. \\
9212   That~key~will~be~ignored. \\
9213   \c_@@_available_keys_str
9214 }
9215 {
9216   The~available~keys~are~(in~alphabetic~order):~
9217   b,~
9218   baseline,~
9219   c,~
9220   cell-space-bottom-limit,~
9221   cell-space-limits,~
9222   cell-space-top-limit,~
9223   code-after,~
9224   code-for-first-col,~
9225   code-for-first-row,~
9226   code-for-last-col,~
9227   code-for-last-row,~
9228   colortbl-like,~
9229   columns-width,~
9230   corners,~
9231   create-extra-nodes,~
9232   create-medium-nodes,~
9233   create-large-nodes,~
9234   extra-left-margin,~
9235   extra-right-margin,~

```

```

9236 first-col,~
9237 first-row,~
9238 hlines,~
9239 hvlines,~
9240 hvlines-except-borders,~
9241 last-col,~
9242 last-row,~
9243 left-margin,~
9244 light-syntax,~
9245 name,~
9246 nullify-dots,~
9247 pgf-node-code,~
9248 renew-dots,~
9249 respect-arraystretch,~
9250 right-margin,~
9251 rules~(with~the~subkeys~'color'~and~'width'),~
9252 small,~
9253 t,~
9254 vlines,~
9255 xdots/color,~
9256 xdots/shorten-start,~
9257 xdots/shorten-end,~
9258 xdots/shorten~and~
9259 xdots/line-style.
9260 }

```

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).

```

9261 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
9262 {
9263   Unknown~key.\\
9264   The~key~'\l_keys_key_str'~is~unknown~for~the~
9265   \@@_full_name_env:. \\
9266   That~key~will~be~ignored. \\
9267   \c_@@_available_keys_str
9268 }
9269 {
9270   The~available~keys~are~(in~alphabetic~order):~
9271   b,~
9272   baseline,~
9273   c,~
9274   cell-space-bottom-limit,~
9275   cell-space-limits,~
9276   cell-space-top-limit,~
9277   code-after,~
9278   code-for-first-col,~
9279   code-for-first-row,~
9280   code-for-last-col,~
9281   code-for-last-row,~
9282   colortbl-like,~
9283   columns-type,~
9284   columns-width,~
9285   corners,~
9286   create-extra-nodes,~
9287   create-medium-nodes,~
9288   create-large-nodes,~
9289   extra-left-margin,~
9290   extra-right-margin,~
9291   first-col,~
9292   first-row,~
9293   hlines,~
9294   hvlines,~
9295   hvlines-except-borders,~
9296   l,~

```

```

9297 last-col,~
9298 last-row,~
9299 left-margin,~
9300 light-syntax,~
9301 name,~
9302 nullify-dots,~
9303 pgf-node-code,~
9304 r,~
9305 renew-dots,~
9306 respect-arraystretch,~
9307 right-margin,~
9308 rules~(with~the~subkeys~'color'~and~'width'),~
9309 small,~
9310 t,~
9311 vlines,~
9312 xdots/color,~
9313 xdots/shorten-start,~
9314 xdots/shorten-end,~
9315 xdots/shorten~and~
9316 xdots/line-style.
9317 }
9318 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9319 {
9320   Unknown~key.\\
9321   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9322   \{NiceTabular\}. \\
9323   That~key~will~be~ignored. \\
9324   \c_@@_available_keys_str
9325 }
9326 {
9327   The~available~keys~are~(in~alphabetic~order):~
9328   b,~
9329   baseline,~
9330   c,~
9331   caption,~
9332   cell-space-bottom-limit,~
9333   cell-space-limits,~
9334   cell-space-top-limit,~
9335   code-after,~
9336   code-for-first-col,~
9337   code-for-first-row,~
9338   code-for-last-col,~
9339   code-for-last-row,~
9340   colortbl-like,~
9341   columns-width,~
9342   corners,~
9343   custom-line,~
9344   create-extra-nodes,~
9345   create-medium-nodes,~
9346   create-large-nodes,~
9347   extra-left-margin,~
9348   extra-right-margin,~
9349   first-col,~
9350   first-row,~
9351   hlines,~
9352   hvlines,~
9353   hvlines-except-borders,~
9354   label,~
9355   last-col,~
9356   last-row,~
9357   left-margin,~
9358   light-syntax,~
9359   name,~

```

```

9360 notes~(several~subkeys),~
9361 nullify-dots,~
9362 pgf-node-code,~
9363 renew-dots,~
9364 respect-arraystretch,~
9365 right-margin,~
9366 rounded-corners,~
9367 rules~(with~the~subkeys~'color'~and~'width'),~
9368 short-caption,~
9369 t,~
9370 tabularnote,~
9371 vlines,~
9372 xdots/color,~
9373 xdots/shorten-start,~
9374 xdots/shorten-end,~
9375 xdots/shorten-and~
9376 xdots/line-style.
9377 }

9378 \@@_msg_new:nnn { Duplicate-name }
9379 {
9380   Duplicate-name.\
9381   The-name~'\l_keys_value_tl'~is~already-used-and-you-shouldn't-use~
9382   the-same-environment-name-twice.~You-can-go-on,~but,~
9383   maybe,~you-will-have-incorrect-results-especially~
9384   if~you-use~'columns-width=auto'.~If~you-don't-want~to~see~this~
9385   message~again,~use~the~key~'allow-duplicate-names'~in~
9386   '\token_to_str:N \NiceMatrixOptions'.\
9387   \bool_if:NF \c_@@_messages_for_Overleaf_bool
9388   { For~a~list~of~the~names~already-used,~type~H~<return>. }
9389 }
9390 {
9391   The-names-already-defined-in-this-document-are:~
9392   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9393 }

9394 \@@_msg_new:nn { Option-auto-for-columns-width }
9395 {
9396   Erroneous-use.\
9397   You-can't-give-the-value~'auto'~to~the~key~'columns-width'~here.~
9398   That~key~will~be~ignored.
9399 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	17
6	Command for creation of rectangle nodes	22
7	The options	23
8	Important code used by <code>{NiceArrayWithDelims}</code>	33
9	The <code>\CodeBefore</code>	45
10	The environment <code>{NiceArrayWithDelims}</code>	49
11	We construct the preamble of the array	54
12	The redefinition of <code>\multicolumn</code>	68
13	The environment <code>{NiceMatrix}</code> and its variants	86
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	87
15	After the construction of the array	88
16	We draw the dotted lines	94
17	The actual instructions for drawing the dotted lines with Tikz	106
18	User commands available in the new environments	111
19	The command <code>\line</code> accessible in code-after	116
20	The command <code>\RowStyle</code>	118
21	Colors of cells, rows and columns	120
22	The vertical and horizontal rules	129
23	The key corners	144
24	The environment <code>{NiceMatrixBlock}</code>	146
25	The extra nodes	147
26	The blocks	151
27	How to draw the dotted lines transparently	169
28	Automatic arrays	169
29	The redefinition of the command <code>\dotfill</code>	171
30	The command <code>\diagbox</code>	171

31	The keyword <code>\CodeAfter</code>	173
32	The delimiters in the preamble	174
33	The command <code>\SubMatrix</code>	175
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	183
35	The command <code>\ShowCellNames</code>	186
36	We process the options at package loading	189
37	About the package underscore	190
38	Error messages of the package	191