

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 14, 2024

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`).

The development of the extension `nicematrix` is done on the following GitHub depot:
<https://github.com/fpantigny/nicematrix>

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}
9 \ProvideDocumentCommand{\IfPackageLoadedT}{mm}
10  {\IfPackageLoadedTF{#1}{#2}{}}
11
12 \ProvideDocumentCommand{\IfPackageLoadedF}{mm}
13  {\IfPackageLoadedTF{#1}{#2}}
```

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 6.28c of `nicematrix`, at the date of 2024/08/22.

In the version 2.6a of array, important modifications have been done for the Tagging Project.

```

16 \bool_const:Nn \c_@@_tagging_array_bool
17 { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
18 \bool_const:Nn \c_@@_testphase_table_bool
19 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool
20 }

21 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
22 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
23 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
24 \cs_generate_variant:Nn \@@_error:nn { n e }
25 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
26 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
27 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
28 \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).

```

29 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
30 {
31   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
32     { \msg_new:nnn { nicematrix } { #1 } { #2 \\\ #3 } }
33     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
34 }

```

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

```

35 \cs_new_protected:Npn \@@_error_or_warning:n
36 { \bool_if:NTF \g_@@_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”.

```

37 \bool_new:N \g_@@_messages_for_Overleaf_bool
38 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
39 {
40   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
41   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
42 }

```

```

43 \cs_new_protected:Npn \@@_msg_redirect_name:nn
44 { \msg_redirect_name:nnn { nicematrix } }
45 \cs_new_protected:Npn \@@_gredirect_none:n #1
46 {
47   \group_begin:
48   \globaldefs = 1
49   \@@_msg_redirect_name:nn { #1 } { none }
50   \group_end:
51 }
52 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
53 {
54   \@@_error:n { #1 }
55   \@@_gredirect_none:n { #1 }
56 }
57 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
58 {
59   \@@_warning:n { #1 }
60   \@@_gredirect_none:n { #1 }
61 }

```

We will delete in the future the following lines which are only a security.

```
62 \cs_set:Npn \int_if_zero:NT #1 { \int_compare:nNnT #1 = \c_zero_int }
63 \cs_set:Npn \int_if_zero:NTF #1 { \int_compare:nNnTF #1 = \c_zero_int }
```

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.

```
64 \@@_msg_new:nn { Internal~error }
65 {
66   Potential~problem~when~using~nicematrix.\\
67   The~package~nicematrix~have~detected~a~modification~of~the~
68   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
69   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
70   this~message~again,~load~nicematrix~with:~\token_to_str:N
71   \usepackage[no-test-for-array]{nicematrix}.
72 }
```

```
73 \@@_msg_new:nn { mdwtab~loaded }
74 {
75   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
76   This~error~is~fatal.
77 }
```

```
78 \cs_new_protected:Npn \@@_security_test:n #1
79 {
80   \peek_meaning:NTF \ignorespaces
81   { \@@_security_test_i:w }
82   { \@@_error:n { Internal~error } }
83   #1
84 }
```

```
85 \bool_if:NTF \c_@@_tagging_array_bool
86 {
87   \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
88   {
89     \peek_meaning:NF \textonly@unskip { \@@_error:n { Internal~error } }
90     #1
91   }
92 }
93 {
94   \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
95   {
96     \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
97     #1
98   }
99 }
```

Here, the box `\l_tmpa_box` will be used as sandbox to take our security test.

```

100 \hook_gput_code:nnn { begindocument / end } { . }
101 {
102   \IfPackageLoadedTF { mdwtab }
103     { \@@_fatal:n { mdwtab~loaded } }
104     {
105       \bool_if:NF \g_@@_no_test_for_array_bool
106       {
107         \group_begin:
108         \hbox_set:Nn \l_tmpa_box
109           {
110             \begin { tabular } { c > { \@@_security_test:n } c c }
111             text & & text
112             \end { tabular }
113           }
114         \group_end:
115       }
116     }
117 }

```

3 Collecting options

The following technic allows to create user commands with the ability to put an arbitrary number of `[list of (key=val)]` after the name of the command.

Exemple :

`\@@_collect_options:n { \F } [x=a,y=b] [z=c,t=d] { arg }`
 will be transformed in : `\F{x=a,y=b,z=c,t=d}{arg}`

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,
 the command `\G` takes in an arbitrary number of optional arguments between square brackets.
 Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

118 \cs_new_protected:Npn \@@_collect_options:n #1
119 {
120   \peek_meaning:NTF [
121     { \@@_collect_options:nw { #1 } }
122     { #1 { } }
123 }

```

We use `\NewDocumentCommand` in order to be able to allow nested brackets within the argument between `[` and `]`.

```

124 \NewDocumentCommand \@@_collect_options:nw { m r[] }
125 { \@@_collect_options:nn { #1 } { #2 } }
126
127 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
128 {
129   \peek_meaning:NTF [
130     { \@@_collect_options:nnw { #1 } { #2 } }
131     { #1 { #2 } }
132 }
133
134 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
135 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

4 Technical definitions

The following constants are defined only for efficiency in the tests.

```

136 \tl_const:Nn \c_@@_b_tl { b }
137 \tl_const:Nn \c_@@_c_tl { c }
138 \tl_const:Nn \c_@@_l_tl { l }
139 \tl_const:Nn \c_@@_r_tl { r }
140 \tl_const:Nn \c_@@_all_tl { all }
141 \tl_const:Nn \c_@@_dot_tl { . }
142 \tl_const:Nn \c_@@_default_tl { default }
143 \tl_const:Nn \c_@@_star_tl { * }
144 \str_const:Nn \c_@@_star_str { * }
145 \str_const:Nn \c_@@_r_str { r }
146 \str_const:Nn \c_@@_c_str { c }
147 \str_const:Nn \c_@@_l_str { l }
148 \str_const:Nn \c_@@_R_str { R }
149 \str_const:Nn \c_@@_C_str { C }
150 \str_const:Nn \c_@@_L_str { L }
151 \str_const:Nn \c_@@_j_str { j }
152 \str_const:Nn \c_@@_si_str { si }

```

For efficiency, we define a small variant of `\clist_if_in:Nn(TF)` which is faster when the items of the clists are safe.

```

153 \prg_new_conditional:Npnn \@@_clist_if_in:Nn #1 #2 { T , F , TF }
154 {
155   \exp_args:Ne \tl_if_in:nnTF { ,#1, } { ,#2, }
156   \prg_return_true:
157   \prg_return_false:
158 }
159 \prg_generate_conditional_variant:Nnn \@@_clist_if_in:Nn { N e } { T , F , TF }

```

The following token list will be used for definitions of user commands (with `\NewDocumentCommand`) with an embellishment using an *underscore* (there may be problems because of the catcode of the underscore).

```

160 \tl_new:N \l_@@_argspec_tl
161 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
162 \cs_generate_variant:Nn \str_lowercase:n { o }
163 \cs_generate_variant:Nn \str_set:Nn { N o }
164 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
165 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F , TF }
166 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
167 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
168 \cs_generate_variant:Nn \dim_min:nn { v }
169 \cs_generate_variant:Nn \dim_max:nn { v }

170 \hook_gput_code:nnn { begindocument } { . }
171 {
172   \IfPackageLoadedTF { tikz }
173   {

```

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 `\hook_gput_code:nnn { begindocument } { . }` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

174 \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }

```

```

175     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
176   }
177   {
178     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
179     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
180   }
181 }

```

We test whether the current class is revtex4-1 (deprecated) or revtex4-2 because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date April 2024, the current version revtex4-2 is 4.2f (compatible with booktabs).

```

182 \IfClassLoadedTF { revtex4-1 }
183 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
184 {
185   \IfClassLoadedTF { revtex4-2 }
186   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
187   {

```

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

```

188     \cs_if_exist:NT \rvtx@ifformat@geq
189     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
190     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
191   }
192 }

```

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.

```

193 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
194 {
195   \iow_now:Nn \@mainaux
196   {
197     \ExplSyntaxOn
198     \cs_if_free:NT \pgfsyspdfmark
199     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
200     \ExplSyntaxOff
201   }
202   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
203 }

```

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.

```

204 \ProvideDocumentCommand \iddots { }
205 {
206   \mathinner
207   {
208     \tex_mkern:D 1 mu
209     \box_move_up:nn { 1 pt } { \hbox { . } }
210     \tex_mkern:D 2 mu
211     \box_move_up:nn { 4 pt } { \hbox { . } }
212     \tex_mkern:D 2 mu
213     \box_move_up:nn { 7 pt }
214     { \vbox:n { \kern 7 pt \hbox { . } } }
215     \tex_mkern:D 1 mu
216   }
217 }

```

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.

```

218 \hook_gput_code:nnn { begindocument } { . }
219 {
220   \IfPackageLoadedT { booktabs }
221     { \iow_now:Nn \mainaux \nicematrix@redefine@check@rerun }
222 }
223 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
224 {
225   \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`).

```

226   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
227   {

```

`\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

228     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
229     { \@@_old_pgful@check@rerun { ##1 } { ##2 } }
230   }
231 }

```

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

```

232 \hook_gput_code:nnn { begindocument } { . }
233 {
234   \IfPackageLoadedF { colortbl }
235   {

```

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.

```

236   \cs_set_protected:Npn \CT@arc@ { }
237   \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
238   \cs_set_nopar:Npn \CT@arc@ #1 #2
239   {
240     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
241       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
242   }

```

Idem for `\CT@drs@`.

```

243   \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
244   \cs_set_nopar:Npn \CT@drs@ #1 #2
245   {
246     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
247       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
248   }
249   \cs_set_nopar:Npn \hline
250   {
251     \noalign { \ifnum 0 = ` } \fi
252     \cs_set_eq:NN \hskip \vskip
253     \cs_set_eq:NN \vrule \hrule
254     \cs_set_eq:NN \@width \@height
255     { \CT@arc@ \vline }
256     \futurelet \reserved@a
257     \@xhline
258   }
259 }
260 }

```

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.

```

261 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }

```

```

262 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
263 {
264   \int_if_zero:nT \l_@@_first_col_int { \omit & }
265   \int_compare:nNnT { #1 } > \c_one_int
266     { \multispan { \int_eval:n { #1 - 1 } } & }
267   \multispan { \int_eval:n { #2 - #1 + 1 } }
268   {
269     \CT@arc@
270     \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

271     \skip_horizontal:N \c_zero_dim
272   }

```

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`.

```

273   \everycr { }
274   \cr
275   \noalign { \skip_vertical:N -\arrayrulewidth }
276 }

```

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.

```

277 \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`.

```

278 { \@@_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*.

```

279 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
280 \cs_generate_variant:Nn \@@_cline_i:nn { e }
281 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
282 {
283   \tl_if_empty:nTF { #3 }
284     { \@@_cline_iii:w #1|#2-#2 \q_stop }
285     { \@@_cline_ii:w #1|#2-#3 \q_stop }
286 }
287 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
288 { \@@_cline_iii:w #1|#2-#3 \q_stop }
289 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
290 {

```

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).

```

291   \int_compare:nNnT { #1 } < { #2 }
292     { \multispan { \int_eval:n { #2 - #1 } } & }
293   \multispan { \int_eval:n { #3 - #2 + 1 } }
294   {
295     \CT@arc@
296     \leaders \hrule \@height \arrayrulewidth \hfill
297     \skip_horizontal:N \c_zero_dim
298   }

```

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

```

299   \peek_meaning_remove_ignore_spaces:NTF \cline
300     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }

```

¹See question 99041 on TeX StackExchange.


```

301     { \everycr { } \cr }
302 }

```

The following command will be nullified in the environment {NiceTabular}, {NiceTabular*} and {NiceTabularX}.

```

303 \cs_set_eq:NN \@_math_toggle: \c_math_toggle_token

304 \cs_generate_variant:Nn \@_set_CT@arc@:n { o }
305 \cs_new_protected:Npn \@_set_CT@arc@:n #1
306 {
307     \tl_if_blank:nF { #1 }
308     {
309         \tl_if_head_eq_meaning:nNTF { #1 } [
310             { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
311             { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
312         ]
313     }

314 \cs_generate_variant:Nn \@_set_CT@drsc@:n { o }
315 \cs_new_protected:Npn \@_set_CT@drsc@:n #1
316 {
317     \tl_if_head_eq_meaning:nNTF { #1 } [
318         { \cs_set_nopar:Npn \CT@drsc@ { \color #1 } }
319         { \cs_set_nopar:Npn \CT@drsc@ { \color { #1 } } }
320     ]

```

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

```

321 \cs_generate_variant:Nn \@_exp_color_arg:Nn { N o }
322 \cs_new:Npn \@_exp_color_arg:Nn #1 #2
323 {
324     \tl_if_head_eq_meaning:nNTF { #2 } [
325         { #1 #2 }
326         { #1 { #2 } }
327     ]

```

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

```

328 \cs_generate_variant:Nn \@_color:n { o }
329 \cs_new_protected:Npn \@_color:n #1
330 { \tl_if_blank:nF { #1 } { \@_exp_color_arg:Nn \color { #1 } } }

331 \cs_new_protected:Npn \@_rescan_for_spanish:N #1
332 {
333     \tl_set_rescan:Nno
334     #1
335     {
336         \char_set_catcode_other:N >
337         \char_set_catcode_other:N <
338     }
339     #1
340 }

```

5 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.

```
341 \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).

```
342 \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.

```
343 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
344 { \int_use:N \g_@@_env_int }
```

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

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

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

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

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

```
348 \bool_new:N \g_@@_delims_bool
349 \bool_gset_true:N \g_@@_delims_bool
```

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

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

```
350 \bool_new:N \l_@@_preamble_bool
351 \bool_set_true:N \l_@@_preamble_bool
```

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

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

The following counter will count the environments `{NiceMatrixBlock}`.

```
353 \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.

```
354 \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).

```
355 \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.).

```
356 \dim_new:N \l_@@_col_width_dim
357 \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.

```
358 \int_new:N \g_@@_row_total_int
359 \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).

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

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

```
361 \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.

```
362 \tl_new:N \l_@@_hpos_cell_tl
363 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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`.

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

Idem for the mono-row blocks.

```
365 \dim_new:N \g_@@_blocks_ht_dim
366 \dim_new:N \g_@@_blocks_dp_dim
```

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

```
367 \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`.

```
368 \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.

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

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

```
370 \bool_new:N \l_@@_notes_detect_duplicates_bool
371 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

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

```
372 \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).

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

The key `color` in a command of rule such as `\Hline` (or the specifier “|” in the preamble of an environment).

```
374 \tl_new:N \l_@@_rule_color_tl
```

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

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

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

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

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

```
377 \bool_new:N \l_@@_X_bool
```

```
378 \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 }`).

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

During the second run, if informations concerning the current environment has been found in the `aux` file, the following flag will be raised.

```
380 \bool_new:N \g_@@_aux_found_bool
```

In particular, in that `aux` file, there will be, for each environment of `nicematrix`, an affectation for the following sequence that will contain informations about the size of the array.

```
381 \seq_new:N \g_@@_size_seq
```

```
382 \tl_new:N \g_@@_left_delim_tl
```

```
383 \tl_new:N \g_@@_right_delim_tl
```

The token list `\g_@@_user_preamble_tl` will contain the preamble provided by the final user of `nicematrix` (eg the preamble of an environment `{NiceTabular}`).

```
384 \tl_new:N \g_@@_user_preamble_tl
```

The token list `\g_@@_array_preamble_tl` will contain the preamble constructed by `nicematrix` for the environment `{array}` (of `array`).

```
385 \tl_new:N \g_@@_array_preamble_tl
```

For `\multicolumn`.

```
386 \tl_new:N \g_@@_preamble_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`.

```
387 \tl_new:N \l_@@_columns_type_tl
```

```
388 \str_set:Nn \l_@@_columns_type_tl { c }
```

The following parameters correspond to the keys `down`, `up` and `middle` of a command such as `\Cdots`. Usually, the final user doesn't use that keys directly because he uses the syntax with the embellishments `_`, `^` and `.`.

```
389 \tl_new:N \l_@@_xdots_down_tl
```

```
390 \tl_new:N \l_@@_xdots_up_tl
```

```
391 \tl_new:N \l_@@_xdots_middle_tl
```

We will store in the following sequence informations provided by the instructions `\rowlistcolors` in the main array (not in the `\CodeBefore`).

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

393 \cs_new_protected:Npn \@@_test_if_math_mode:
394 {
395   \if_mode_math: \else:
396     \@@_fatal:n { Outside-math-mode }
397   \fi:
398 }
```

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.

```
399 \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”.

```
400 \colorlet { nicematrix-last-col } { . }
401 \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`).

```
402 \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*.

```
403 \tl_new:N \g_@@_com_or_env_str
404 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
405 \bool_new:N \l_@@_bold_row_style_bool
```

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:eeTF` and not `\tl_if_eq:eeTF` because we need to be fully expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
406 \cs_new:Npn \@@_full_name_env:
407 {
408   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
409     { command \space \c_backslash_str \g_@@_name_env_str }
410     { environment \space \{ \g_@@_name_env_str \} }
411 }
```

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

```
412 \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.

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

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

```
414 \tl_new:N \g_@@_pre_code_before_tl
415 \tl_new:N \g_nicematrix_code_before_tl
```

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

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

```
416 \tl_new:N \g_@@_pre_code_after_tl
417 \tl_new:N \g_nicematrix_code_after_tl
```

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

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

The following parameter will be raised when a block content a `&` in its content (=label).

```
419 \bool_new:N \l_@@_ampersand_bool
```

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

```
420 \int_new:N \l_@@_old_iRow_int
421 \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).

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

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

```
423 \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.

```
424 \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.

```
425 \bool_new:N \l_@@_X_columns_aux_bool
426 \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.

```
427 \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).

```
428 \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).

```
429 \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`).

```
430 \tl_new:N \l_@@_code_before_tl
431 \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).

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

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

```
433 \dim_new:N \l_@@_x_initial_dim
434 \dim_new:N \l_@@_y_initial_dim
435 \dim_new:N \l_@@_x_final_dim
436 \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.

```
437 \dim_new:N \l_@@_tmpc_dim
438 \dim_new:N \l_@@_tmpd_dim

439 \dim_new:N \g_@@_dp_row_zero_dim
440 \dim_new:N \g_@@_ht_row_zero_dim
441 \dim_new:N \g_@@_ht_row_one_dim
442 \dim_new:N \g_@@_dp_ante_last_row_dim
443 \dim_new:N \g_@@_ht_last_row_dim
444 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
445 \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”.

```
446 \dim_new:N \g_@@_width_last_col_dim
447 \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.

```
448 \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!).

```
449 \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}`.

```
450 \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.

```
451 \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 list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
452 \clist_new:N \l_@@_corners_cells_clist
```

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`).

```
453 \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.

```
454 \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).

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

```
456 \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`).

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

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

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

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

The following counters will be used when drawing the rules.

```
461 \int_new:N \l_@@_start_int
```

```
462 \int_set_eq:NN \l_@@_start_int \c_one_int
```

```
463 \int_new:N \l_@@_end_int
```

```
464 \int_new:N \l_@@_local_start_int
```

```
465 \int_new:N \l_@@_local_end_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.

```
466 \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).

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


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

```
468 \tl_new:N \l_@@_fill_tl
469 \tl_new:N \l_@@_opacity_tl
470 \tl_new:N \l_@@_draw_tl
471 \seq_new:N \l_@@_tikz_seq
472 \clist_new:N \l_@@_borders_clist
473 \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.

```
474 \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`.

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

In the key `tikz` of a command `\Block` or in the argument of a command `\TikzEveryCell`, the final user puts a list of `tikz` keys. But, you have added another key, named `offset` (which means that an offset will be used for the frame of the block or the cell). The following parameter corresponds to that key.

```
476 \dim_new:N \l_@@_offset_dim
```

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

```
477 \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`).

```
478 \str_new:N \l_@@_hpos_block_str
479 \str_set:Nn \l_@@_hpos_block_str { c }
480 \bool_new:N \l_@@_hpos_of_block_cap_bool
481 \bool_new:N \l_@@_p_block_bool
```

If the final user has used the special color “`nocolor`”, the following flag will be raised.

```
482 \bool_new:N \l_@@_nocolor_used_bool
```

For the vertical position, the possible values are `c`, `t`, `b`, `T` and `B` (but `\l_@@_vpos_block_str` will remain empty if the user doesn’t use a key for the vertical position).

```
483 \str_new:N \l_@@_vpos_block_str
```

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

```
484 \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).

```
485 \bool_new:N \l_@@_vlines_block_bool
486 \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.

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

```

488 \dim_new:N \l_@@_submatrix_extra_height_dim
489 \dim_new:N \l_@@_submatrix_left_xshift_dim
490 \dim_new:N \l_@@_submatrix_right_xshift_dim
491 \clist_new:N \l_@@_hlines_clist
492 \clist_new:N \l_@@_vlines_clist
493 \clist_new:N \l_@@_submatrix_hlines_clist
494 \clist_new:N \l_@@_submatrix_vlines_clist

```

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

```

495 \bool_new:N \l_@@_hlines_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.

```

496 \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`).

```

497 \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.

```

498 \int_new:N \l_@@_first_row_int
499 \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.

```

500 \int_new:N \l_@@_first_col_int
501 \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

• 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).

```

502 \int_new:N \l_@@_last_row_int
503 \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

```

504 \bool_new:N \l_@@_last_row_without_value_bool

```

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

Idem for `\l_@@_last_col_without_value_bool`

```
505 \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. The command `\NiceMatrixOptions` also sets `\l_@@_last_col_int` to 0 .

```
506 \int_new:N \l_@@_last_col_int
507 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

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

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

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

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

Some utilities

```
510 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
511 {
```

Here, we use `\cs_set_nopar:Npn` instead of `\tl_set:Nn` for efficiency only.

```
512 \cs_set_nopar:Npn \l_tmpa_tl { #1 }
513 \cs_set_nopar:Npn \l_tmpb_tl { #2 }
514 }
```

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.

```
515 \cs_new_protected:Npn \@@_expand_clist:N #1
516 {
```

`\@@_clist_if_in:NnF` is only a variant of `\clist_if_in:NnF` which is faster when the items in the `clists` are safe.

```
517 \@@_clist_if_in:NnF #1 { all }
518 {
519 \clist_clear:N \l_tmpa_clist
520 \clist_map_inline:Nn #1
521 {
```

We recall thant `\tl_if_in:nnTF` is slightly faster than `\str_if_in:nnTF`.

```
522 \tl_if_in:nnTF { ##1 } { - }
523 { \@@_cut_on_hyphen:w ##1 \q_stop }
524 {
```

Here, we use `\cs_set_nopar:Npn` instead of `\tl_set:Nn` for efficiency only.

```

525         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
526         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
527     }
528     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
529     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
530 }
531 \tl_set_eq:NN #1 \l_tmpa_clist
532 }
533 }
```

The following internal parameters are for:

- `\Ldots` with both extremities open (and hence also `\Hdotsfor` in an exterior row;
- `\Vdots` with both extremities open (and hence also `\Vdotsfor` in an exterior column;
- when the special character “:” is used in order to put the label of a so-called “dotted line” *on the line*, a margin of `\c_@@_innersep_middle_dim` will be added around the label.

```

534 \hook_gput_code:nnn { begindocument } { . }
535 {
536     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
537     \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
538     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
539 }
```

6 The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It’s also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that’s mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That’s why:
 - The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.³
 - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component 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 expressed by `\c_novalue_tl`).

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

- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

The LaTeX counter `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.

```
540 \newcounter { tabularnote }
```

We want to avoid error messages for duplicate labels when the package `hyperref` is used. That's why we will count all the tabular notes of the whole document with `\g_@@_tabularnote_int`.

```
541 \int_new:N \g_@@_tabularnote_int
542 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
543 \seq_new:N \g_@@_notes_seq
544 \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.

```
545 \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.

```
546 \seq_new:N \l_@@_notes_labels_seq
547 \newcounter{nicematrix_draft}
548 \cs_new_protected:Npn \@@_notes_format:n #1
549 {
550   \setcounter { nicematrix_draft } { #1 }
551   \@@_notes_style:n { nicematrix_draft }
552 }
```

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

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

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

```
554 \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`.

```
555 \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`.

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

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

```
557 \hook_gput_code:nnn { begindocument } { . }
558 {
559   \IfPackageLoadedTF { enumitem }
560   {
```

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.

```

561     \newlist { tabularnotes } { enumerate } { 1 }
562     \setlist [ tabularnotes ]
563     {
564         topsep = Opt ,
565         noitemsep ,
566         leftmargin = * ,
567         align = left ,
568         labelsep = Opt ,
569         label =
570             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
571     }
572     \newlist { tabularnotes* } { enumerate* } { 1 }
573     \setlist [ tabularnotes* ]
574     {
575         afterlabel = \nobreak ,
576         itemjoin = \quad ,
577         label =
578             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
579     }

```

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`.

```

580     \NewDocumentCommand \tabularnote { o m }
581     {
582         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } \l_@@_in_env_bool
583         {
584             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
585             { \@@_error:n { tabularnote~forbidden } }
586             {
587                 \bool_if:NTF \l_@@_in_caption_bool
588                 \@@_tabularnote_caption:nn
589                 \@@_tabularnote:nn
590                 { #1 } { #2 }
591             }
592         }
593     }
594 }
595 {
596     \NewDocumentCommand \tabularnote { o m }
597     {
598         \@@_error_or_warning:n { enumitem~not~loaded }
599         \@@_gredirect_none:n { enumitem~not~loaded }
600     }
601 }
602 }
603 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
604 { \tl_if_novalue:nT { #1 } { #3 } }

```

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 expressed by `\c_novalue_tl`) and `#2` is the mandatory argument of `\tabularnote`.

```

605 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
606 {

```

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).

```

607     \int_zero:N \l_tmpa_int
608     \bool_if:NT \l_@@_notes_detect_duplicates_bool
609     {

```

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

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

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

When we will go through the sequence `\g_@@_notes_seq`, we will count in `\l_tmpb_int` the notes without explicit label in order to have the “current” value of the counter `\c@tabulernote`.

```

610     \int_zero:N \l_tmpb_int
611     \seq_map_indexed_inline:Nn \g_@@_notes_seq
612     {
613         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
614         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
615             {
616                 \tl_if_novalue:nTF { #1 }
617                     { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
618                     { \int_set:Nn \l_tmpa_int { ##1 } }
619                 \seq_map_break:
620             }
621         }
622         \int_if_zero:nF \l_tmpa_int
623         { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
624     }
625     \int_if_zero:nT \l_tmpa_int
626     {
627         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
628         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
629     }
630     \seq_put_right:Ne \l_@@_notes_labels_seq
631     {
632         \tl_if_novalue:nTF { #1 }
633         {
634             \@@_notes_format:n
635             {
636                 \int_eval:n
637                 {
638                     \int_if_zero:nTF \l_tmpa_int
639                     \c@tabulernote
640                     \l_tmpa_int
641                 }
642             }
643         }
644         { #1 }
645     }
646     \peek_meaning:NF \tabulernote
647     {

```

If the following token is *not* a `\tabulernote`, we have finished the sequence of successive commands `\tabulernote` 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_tl` is equal to `c` or `r`.

```

648     \hbox_set:Nn \l_tmpa_box
649     {

```

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

```

650         \@@_notes_label_in_tabular:n
651         {

```

```

652         \seq_use:Nnnn
653         \l_@@_notes_labels_seq { , } { , } { , }
654     }
655 }

```

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

```

656     \int_gdecr:N \c@tabularnote
657     \int_set_eq:NN \l_tmpa_int \c@tabularnote

```

The following line is only to avoid error messages for multiply defined labels when the package `hyperref` is used.

```

658     \int_gincr:N \g_@@_tabularnote_int
659     \refstepcounter { tabularnote }
660     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
661     { \int_gincr:N \c@tabularnote }
662     \seq_clear:N \l_@@_notes_labels_seq
663     \bool_lazy_or:nnTF
664     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
665     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
666     {
667         \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).

```

668         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
669     }
670     { \box_use:N \l_tmpa_box }
671 }
672 }

```

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`.

```

673 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
674 {
675     \bool_if:NTF \g_@@_caption_finished_bool
676     {
677         \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
678         { \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`!

```

679         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
680         { \@@_error:n { Identical~notes~in~caption } }
681     }
682     {

```

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

```

683         \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
684         {

```

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.

```

685         \bool_gset_true:N \g_@@_caption_finished_bool
686         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
687         \int_gzero:N \c@tabularnote
688     }
689     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
690 }

```


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!

```

691 \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
692 \seq_put_right:Ne \l_@@_notes_labels_seq
693 {
694   \tl_if_novalue:nTF { #1 }
695     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
696     { #1 }
697 }
698 \peek_meaning:NF \tabularnote
699 {
700   \@@_notes_label_in_tabular:n
701     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
702   \seq_clear:N \l_@@_notes_labels_seq
703 }
704 }

705 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
706 { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }

```

7 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.

```

707 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
708 {
709   \begin { pgfscope }
710   \pgfset
711     {
712       inner~sep = \c_zero_dim ,
713       minimum~size = \c_zero_dim
714     }
715   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
716   \pgfnode
717     { rectangle }
718     { center }
719     {
720       \vbox_to_ht:nn
721         { \dim_abs:n { #5 - #3 } }
722         {
723           \vfill
724           \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
725         }
726     }
727     { #1 }
728     { }
729   \end { pgfscope }
730 }

```

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.

```

731 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
732 {
733   \begin { pgfscope }
734   \pgfset
735     {
736       inner~sep = \c_zero_dim ,
737       minimum~size = \c_zero_dim

```

```

738     }
739     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
740     \pgfpointdiff { #3 } { #2 }
741     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
742     \pgfnode
743     { rectangle }
744     { center }
745     {
746         \vbox_to_ht:nn
747         { \dim_abs:n \l_tmpb_dim }
748         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
749     }
750     { #1 }
751     { }
752 \end { pgfscope }
753 }

```

8 The options

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

```

754 \tl_new:N \l_@@_caption_tl
755 \tl_new:N \l_@@_short_caption_tl
756 \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).

```

757 \bool_new:N \l_@@_caption_above_bool

```

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

```

758 \bool_new:N \l_@@_color_inside_bool

```

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

```

759 \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`).

```

760 \dim_new:N \l_@@_cell_space_top_limit_dim
761 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

762 \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.

```

763 \dim_new:N \l_@@_xdots_inter_dim
764 \hook_gput_code:nnn { begindocument } { . }
765 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

The unit is `em` and that's why we fix the dimension after the preamble.

The following dimension is the distance between a node (in fact an anchor of that node) and a dotted line (for real dotted lines, the actual distance may, of course, be a bit larger, depending of the exact position of the dots).

```

766 \dim_new:N \l_@@_xdots_shorten_start_dim
767 \dim_new:N \l_@@_xdots_shorten_end_dim
768 \hook_gput_code:nnn { begindocument } { . }
769 {
770   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
771   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
772 }
```

The unit is `em` and that's why we fix the dimension after the preamble.

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.

```

773 \dim_new:N \l_@@_xdots_radius_dim
774 \hook_gput_code:nnn { begindocument } { . }
775 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

The unit is `em` and that's why we fix the dimension after the preamble.

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.

```

776 \tl_new:N \l_@@_xdots_line_style_tl
777 \tl_const:Nn \c_@@_standard_tl { standard }
778 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax` and the boolean `\l_@@_light_syntax_expanded_bool` correspond to the the option `light-syntax-expanded`.

```

779 \bool_new:N \l_@@_light_syntax_bool
780 \bool_new:N \l_@@_light_syntax_expanded_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).

```

781 \tl_new:N \l_@@_baseline_tl
782 \tl_set:Nn \l_@@_baseline_tl { c }
```

The following parameter corresponds to the key `ampersand-in-blocks`

```

783 \bool_new:N \l_@@_amp_in_blocks_bool
```

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`).

```

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

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

```

785 \bool_new:N \l_@@_parallelize_diags_bool
786 \bool_set_true:N \l_@@_parallelize_diags_bool
```

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

```

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

```

788 \dim_new:N \l_@@_notes_above_space_dim
789 \hook_gput_code:nnn { begindocument } { . }
790 { \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.).

```

791 \bool_new:N \l_@@_nullify_dots_bool

```

When the key `respect-arraystretch` is used, the following command will be nullified.

```

792 \cs_new_protected:Npn \@@_reset_arraystretch:
793 { \cs_set_nopar:Npn \arraystretch { 1 } }

```

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).

```

794 \bool_new:N \l_@@_auto_columns_width_bool

```

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

```

795 \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.

```

796 \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”.

```

797 \bool_new:N \l_@@_medium_nodes_bool
798 \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).

```

799 \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`).

```

800 \dim_new:N \l_@@_left_margin_dim
801 \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`.

```

802 \dim_new:N \l_@@_extra_left_margin_dim
803 \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.

```

804 \tl_new:N \l_@@_end_of_row_tl
805 \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 “:”.

```

806 \tl_new:N \l_@@_xdots_color_tl

```

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

```
807 \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.

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

```
809 \keys_define:nn { nicematrix / xdots }
810 {
811   shorten-start .code:n =
812     \hook_gput_code:nnn { begindocument } { . }
813     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
814   shorten-end .code:n =
815     \hook_gput_code:nnn { begindocument } { . }
816     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
817   shorten-start .value_required:n = true ,
818   shorten-end .value_required:n = true ,
819   shorten .code:n =
820     \hook_gput_code:nnn { begindocument } { . }
821     {
822       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
823       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
824     } ,
825   shorten .value_required:n = true ,
826   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
827   horizontal-labels .default:n = true ,
828   line-style .code:n =
829     {
830       \bool_lazy_or:nnTF
831         { \cs_if_exist_p:N \tikzpicture }
832         { \str_if_eq_p:nn { #1 } { standard } }
833         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
834         { \@@_error:n { bad-option-for-line-style } }
835     } ,
836   line-style .value_required:n = true ,
837   color .tl_set:N = \l_@@_xdots_color_tl ,
838   color .value_required:n = true ,
839   radius .code:n =
840     \hook_gput_code:nnn { begindocument } { . }
841     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
842   radius .value_required:n = true ,
843   inter .code:n =
844     \hook_gput_code:nnn { begindocument } { . }
845     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
846   radius .value_required:n = true ,
```

The options `down`, `up` and `middle` are not documented for the final user because he should use the syntax with `^`, `_` and `:`. We use `\tl_put_right:Nn` and not `\tl_set:Nn` (or `.tl_set:N`) because we don't want a direct use of `up=...` erased by an absent `^{\dots}`.

```
847   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
848   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
849   middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,
```

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

```
850   draw-first .code:n = \prg_do_nothing: ,
```

```

851     unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
852 }

853 \keys_define:nn { nicematrix / rules }
854 {
855     color .tl_set:N = \l_@@_rules_color_tl ,
856     color .value_required:n = true ,
857     width .dim_set:N = \arrayrulewidth ,
858     width .value_required:n = true ,
859     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
860 }

```

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

```

861 \keys_define:nn { nicematrix / Global }
862 {
863     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
864     ampersand-in-blocks .default:n = true ,
865     &-in-blocks .meta:n = ampersand-in-blocks ,
866     no-cell-nodes .code:n =
867         \cs_set_protected:Npn \@@_node_for_cell:
868         { \box_use_drop:N \l_@@_cell_box } ,
869     no-cell-nodes .value_forbidden:n = true ,
870     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
871     rounded-corners .default:n = 4 pt ,
872     custom-line .code:n = \@@_custom_line:n { #1 } ,
873     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
874     rules .value_required:n = true ,
875     standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
876     standard-cline .default:n = true ,
877     cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
878     cell-space-top-limit .value_required:n = true ,
879     cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
880     cell-space-bottom-limit .value_required:n = true ,
881     cell-space-limits .meta:n =
882     {
883         cell-space-top-limit = #1 ,
884         cell-space-bottom-limit = #1 ,
885     } ,
886     cell-space-limits .value_required:n = true ,
887     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
888     light-syntax .code:n =
889         \bool_set_true:N \l_@@_light_syntax_bool
890         \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
891     light-syntax .value_forbidden:n = true ,
892     light-syntax-expanded .code:n =
893         \bool_set_true:N \l_@@_light_syntax_bool
894         \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
895     light-syntax-expanded .value_forbidden:n = true ,
896     end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
897     end-of-row .value_required:n = true ,
898     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
899     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
900     last-row .int_set:N = \l_@@_last_row_int ,
901     last-row .default:n = -1 ,
902     code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
903     code-for-first-col .value_required:n = true ,
904     code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
905     code-for-last-col .value_required:n = true ,
906     code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
907     code-for-first-row .value_required:n = true ,
908     code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,

```

```

909 code-for-last-row .value_required:n = true ,
910 hlines .clist_set:N = \l_@@_hlines_clist ,
911 vlines .clist_set:N = \l_@@_vlines_clist ,
912 hlines .default:n = all ,
913 vlines .default:n = all ,
914 vlines-in-sub-matrix .code:n =
915 {
916   \tl_if_single_token:nTF { #1 }
917   {
918     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
919     { \@@_error:nn { Forbidden-letter } { #1 } }

```

We write directly a command for the automata which reads the preamble provided by the final user.

```

920     { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
921   }
922   { \@@_error:n { One-letter-allowed } }
923 } ,
924 vlines-in-sub-matrix .value_required:n = true ,
925 hvlines .code:n =
926 {
927   \bool_set_true:N \l_@@_hvlines_bool
928   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
929   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
930 } ,
931 hvlines-except-borders .code:n =
932 {
933   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
934   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
935   \bool_set_true:N \l_@@_hvlines_bool
936   \bool_set_true:N \l_@@_except_borders_bool
937 } ,
938 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.

```

939 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
940 renew-dots .value_forbidden:n = true ,
941 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
942 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
943 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
944 create-extra-nodes .meta:n =
945 { create-medium-nodes , create-large-nodes } ,
946 left-margin .dim_set:N = \l_@@_left_margin_dim ,
947 left-margin .default:n = \arraycolsep ,
948 right-margin .dim_set:N = \l_@@_right_margin_dim ,
949 right-margin .default:n = \arraycolsep ,
950 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
951 margin .default:n = \arraycolsep ,
952 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
953 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
954 extra-margin .meta:n =
955 { extra-left-margin = #1 , extra-right-margin = #1 } ,
956 extra-margin .value_required:n = true ,
957 respect-arraystretch .code:n =
958   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
959 respect-arraystretch .value_forbidden:n = true ,
960 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
961 pgf-node-code .value_required:n = true
962 }

```

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

```

963 \keys_define:nn { nicematrix / environments }
964 {
965   corners .clist_set:N = \l_@@_corners_clist ,
966   corners .default:n = { NW , SW , NE , SE } ,
967   code-before .code:n =
968   {
969     \tl_if_empty:nF { #1 }
970     {
971       \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
972       \bool_set_true:N \l_@@_code_before_bool
973     }
974   } ,
975   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}`.

```

976   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
977   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
978   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
979   baseline .tl_set:N = \l_@@_baseline_tl ,
980   baseline .value_required:n = true ,
981   columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

982   \str_if_eq:eeTF { #1 } { auto }
983   { \bool_set_true:N \l_@@_auto_columns_width_bool }
984   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
985   columns-width .value_required:n = true ,
986   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.

```

987   \legacy_if:nF { measuring@ }
988   {
989     \str_set:Ne \l_tmpa_str { #1 }
990     \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
991     { \@@_error:nn { Duplicate-name } { #1 } }
992     { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
993     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
994   } ,
995   name .value_required:n = true ,
996   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
997   code-after .value_required:n = true ,
998   color-inside .code:n =
999   \bool_set_true:N \l_@@_color_inside_bool
1000   \bool_set_true:N \l_@@_code_before_bool ,
1001   color-inside .value_forbidden:n = true ,
1002   colortbl-like .meta:n = color-inside
1003 }

1004 \keys_define:nn { nicematrix / notes }
1005 {
1006   para .bool_set:N = \l_@@_notes_para_bool ,
1007   para .default:n = true ,
1008   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1009   code-before .value_required:n = true ,
1010   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1011   code-after .value_required:n = true ,
1012   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1013   bottomrule .default:n = true ,
1014   style .cs_set:Np = \@@_notes_style:n #1 ,
1015   style .value_required:n = true ,
1016   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,

```



```

1017 label-in-tabular .value_required:n = true ,
1018 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1019 label-in-list .value_required:n = true ,
1020 enumitem-keys .code:n =
1021 {
1022   \hook_gput_code:nnn { begindocument } { . }
1023   {
1024     \IfPackageLoadedT { enumitem }
1025     { \setlist* [ tabularnotes ] { #1 } }
1026   }
1027 } ,
1028 enumitem-keys .value_required:n = true ,
1029 enumitem-keys-para .code:n =
1030 {
1031   \hook_gput_code:nnn { begindocument } { . }
1032   {
1033     \IfPackageLoadedT { enumitem }
1034     { \setlist* [ tabularnotes* ] { #1 } }
1035   }
1036 } ,
1037 enumitem-keys-para .value_required:n = true ,
1038 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1039 detect-duplicates .default:n = true ,
1040 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1041 }
1042 \keys_define:nn { nicematrix / delimiters }
1043 {
1044   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1045   max-width .default:n = true ,
1046   color .tl_set:N = \l_@@_delimiters_color_tl ,
1047   color .value_required:n = true ,
1048 }

```

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

```

1049 \keys_define:nn { nicematrix }
1050 {
1051   NiceMatrixOptions .inherit:n =
1052   { nicematrix / Global } ,
1053   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1054   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1055   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1056   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1057   SubMatrix / rules .inherit:n = nicematrix / rules ,
1058   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1059   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1060   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1061   NiceMatrix .inherit:n =
1062   {
1063     nicematrix / Global ,
1064     nicematrix / environments ,
1065   } ,
1066   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1067   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1068   NiceTabular .inherit:n =
1069   {
1070     nicematrix / Global ,
1071     nicematrix / environments
1072   } ,
1073   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1074   NiceTabular / rules .inherit:n = nicematrix / rules ,
1075   NiceTabular / notes .inherit:n = nicematrix / notes ,

```

```

1076   NiceArray .inherit:n =
1077   {
1078       nicematrix / Global ,
1079       nicematrix / environments ,
1080   } ,
1081   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1082   NiceArray / rules .inherit:n = nicematrix / rules ,
1083   pNiceArray .inherit:n =
1084   {
1085       nicematrix / Global ,
1086       nicematrix / environments ,
1087   } ,
1088   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1089   pNiceArray / rules .inherit:n = nicematrix / rules ,
1090 }

```

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

```

1091 \keys_define:nn { nicematrix / NiceMatrixOptions }
1092 {
1093     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1094     delimiters / color .value_required:n = true ,
1095     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1096     delimiters / max-width .default:n = true ,
1097     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1098     delimiters .value_required:n = true ,
1099     width .dim_set:N = \l_@@_width_dim ,
1100     width .value_required:n = true ,
1101     last-col .code:n =
1102         \tl_if_empty:nF { #1 }
1103         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1104         \int_zero:N \l_@@_last_col_int ,
1105     small .bool_set:N = \l_@@_small_bool ,
1106     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.

```

1107     renew-matrix .code:n = \@@_renew_matrix: ,
1108     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}`.

```

1109     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.

```

1110     columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1111     \str_if_eq:eeTF { #1 } { auto }
1112     { \@@_error:n { Option~auto~for~columns-width } }
1113     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1114     allow-duplicate-names .code:n =
1115         \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1116     allow-duplicate-names .value_forbidden:n = true ,
1117     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,

```

```

1118     notes .value_required:n = true ,
1119     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1120     sub-matrix .value_required:n = true ,
1121     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1122     matrix / columns-type .value_required:n = true ,
1123     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1124     caption-above .default:n = true ,
1125     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1126 }

```

`\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.

```

1127 \NewDocumentCommand \NiceMatrixOptions { m }
1128 { \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.

```

1129 \keys_define:nn { nicematrix / NiceMatrix }
1130 {
1131     last-col .code:n = \tl_if_empty:nTF { #1 }
1132     {
1133         \bool_set_true:N \l_@@_last_col_without_value_bool
1134         \int_set:Nn \l_@@_last_col_int { -1 }
1135     }
1136     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1137     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1138     columns-type .value_required:n = true ,
1139     l .meta:n = { columns-type = l } ,
1140     r .meta:n = { columns-type = r } ,
1141     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1142     delimiters / color .value_required:n = true ,
1143     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1144     delimiters / max-width .default:n = true ,
1145     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1146     delimiters .value_required:n = true ,
1147     small .bool_set:N = \l_@@_small_bool ,
1148     small .value_forbidden:n = true ,
1149     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1150 }

```

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

```

1151 \keys_define:nn { nicematrix / NiceArray }
1152 {

```

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.

```

1153     small .bool_set:N = \l_@@_small_bool ,
1154     small .value_forbidden:n = true ,
1155     last-col .code:n = \tl_if_empty:nF { #1 }
1156     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1157     \int_zero:N \l_@@_last_col_int ,
1158     r .code:n = \@@_error:n { r-or~l~with~preamble } ,
1159     l .code:n = \@@_error:n { r-or~l~with~preamble } ,
1160     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1161 }
1162 \keys_define:nn { nicematrix / pNiceArray }
1163 {
1164     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1165     last-col .code:n = \tl_if_empty:nF { #1 }
1166     { \@@_error:n { last-col~non~empty~for~NiceArray } }

```

```

1167         \int_zero:N \l_@@_last_col_int ,
1168     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1169     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1170     delimiters / color .value_required:n = true ,
1171     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1172     delimiters / max-width .default:n = true ,
1173     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1174     delimiters .value_required:n = true ,
1175     small .bool_set:N = \l_@@_small_bool ,
1176     small .value_forbidden:n = true ,
1177     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1178     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1179     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1180 }

```

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

```

1181 \keys_define:nn { nicematrix / NiceTabular }
1182 {

```

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.

```

1183     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1184     \bool_set_true:N \l_@@_width_used_bool ,
1185     width .value_required:n = true ,
1186     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1187     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1188     tabularnote .value_required:n = true ,
1189     caption .tl_set:N = \l_@@_caption_tl ,
1190     caption .value_required:n = true ,
1191     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1192     short-caption .value_required:n = true ,
1193     label .tl_set:N = \l_@@_label_tl ,
1194     label .value_required:n = true ,
1195     last-col .code:n = \tl_if_empty:nF { #1 }
1196         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1197     \int_zero:N \l_@@_last_col_int ,
1198     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1199     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1200     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1201 }

```

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. We *must* put the following instructions *after* the :

`CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix`

```

1202 \keys_define:nn { nicematrix / CodeAfter }
1203 {
1204     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1205     delimiters / color .value_required:n = true ,
1206     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1207     rules .value_required:n = true ,
1208     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1209     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1210     sub-matrix .value_required:n = true ,
1211     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1212 }

```

9 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}`).

```
1213 \cs_new_protected:Npn \@@_cell_begin:w
1214 {
```

`\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.

```
1215 \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).

```
1216 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

We increment the LaTeX counter `jCol`, which is the counter of the columns.

```
1217 \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.

```
1218 \int_compare:nNnT \c@jCol = \c_one_int
1219 { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_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` is in the `\@@_cell_end:`.

```
1220 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1221 \@@_tuning_not_tabular_begin:
1222 \@@_tuning_first_row:
1223 \@@_tuning_last_row:
1224 \g_@@_row_style_tl
1225 }
```

The following command will be nullified unless there is a first row.

Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT \c@iRow
  {
    \int_compare:nNnT \c@jCol > 0
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}
```

We will use a version a little more efficient.

```
1226 \cs_new_protected:Npn \@@_tuning_first_row:
1227 {
1228   \if_int_compare:w \c@iRow = \c_zero_int
1229   \if_int_compare:w \c@jCol > \c_zero_int
1230   \l_@@_code_for_first_row_tl
1231   \xglobal \colorlet { nicematrix-first-row } { . }
1232   \fi:
1233   \fi:
1234 }
```

The following command will be nullified unless there is a last row and we know its value (*ie*: `\l_@@_lat_row_int > 0`).

```

\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare:nNtT \c@iRow = \l_@@_last_row_int
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}

```

We will use a version a little more efficient.

```

1235 \cs_new_protected:Npn \@@_tuning_last_row:
1236 {
1237   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1238     \l_@@_code_for_last_row_tl
1239     \xglobal \colorlet { nicematrix-last-row } { . }
1240   \fi:
1241 }

```

A different value will be provided to the following command when the key `small` is in force.

```

1242 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1243 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1244 {
1245   \c_math_toggle_token

```

A special value is provided by the following controls sequence when the key `small` is in force.

```

1246   \@@_tuning_key_small:
1247 }
1248 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

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.

```

1249 \cs_new_protected:Npn \@@_begin_of_row:
1250 {
1251   \int_gincr:N \c@iRow
1252   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1253   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1254   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1255   \pgfpicture
1256   \pgfrememberpicturepositiononpagetrue
1257   \pgfcoordinate
1258   { \@@_env: - row - \int_use:N \c@iRow - base }
1259   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1260   \str_if_empty:NF \l_@@_name_str
1261   {
1262     \pgfnodealias
1263     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1264     { \@@_env: - row - \int_use:N \c@iRow - base }
1265   }
1266   \endpgfpicture
1267 }

```

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.

```

1268 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1269 {
1270   \int_if_zero:nTF \c@iRow

```

```

1271 {
1272   \dim_gset:Nn \g_@@_dp_row_zero_dim
1273   { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1274   \dim_gset:Nn \g_@@_ht_row_zero_dim
1275   { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1276 }
1277 {
1278   \int_compare:nNnT \c_iRow = \c_one_int
1279   {
1280     \dim_gset:Nn \g_@@_ht_row_one_dim
1281     { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1282   }
1283 }
1284 }
1285 \cs_new_protected:Npn \@@_rotate_cell_box:
1286 {
1287   \box_rotate:Nn \l_@@_cell_box { 90 }
1288   \bool_if:NTF \g_@@_rotate_c_bool
1289   {
1290     \hbox_set:Nn \l_@@_cell_box
1291     {
1292       \c_math_toggle_token
1293       \vcenter { \box_use:N \l_@@_cell_box }
1294       \c_math_toggle_token
1295     }
1296   }
1297   {
1298     \int_compare:nNnT \c_iRow = \l_@@_last_row_int
1299     {
1300       \vbox_set_top:Nn \l_@@_cell_box
1301       {
1302         \vbox_to_zero:n { }
1303         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1304         \box_use:N \l_@@_cell_box
1305       }
1306     }
1307   }
1308   \bool_gset_false:N \g_@@_rotate_bool
1309   \bool_gset_false:N \g_@@_rotate_c_bool
1310 }
1311 \cs_new_protected:Npn \@@_adjust_size_box:
1312 {
1313   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1314   {
1315     \box_set_wd:Nn \l_@@_cell_box
1316     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1317     \dim_gzero:N \g_@@_blocks_wd_dim
1318   }
1319   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1320   {
1321     \box_set_dp:Nn \l_@@_cell_box
1322     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1323     \dim_gzero:N \g_@@_blocks_dp_dim
1324   }
1325   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1326   {
1327     \box_set_ht:Nn \l_@@_cell_box
1328     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1329     \dim_gzero:N \g_@@_blocks_ht_dim
1330   }
1331 }
1332 \cs_new_protected:Npn \@@_cell_end:
1333 {

```

The following command is nullified in the tabulars.

```

1334 \@@_tuning_not_tabular_end:
1335 \hbox_set_end:
1336 \@@_cell_end_i:
1337 }
1338 \cs_new_protected:Npn \@@_cell_end_i:
1339 {

```

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.

```

1340 \g_@@_cell_after_hook_tl
1341 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1342 \@@_adjust_size_box:
1343 \box_set_ht:Nn \l_@@_cell_box
1344 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1345 \box_set_dp:Nn \l_@@_cell_box
1346 { \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”).

```

1347 \@@_update_max_cell_width:

```

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

```

1348 \@@_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.

```

1349 \bool_if:NTF \g_@@_empty_cell_bool
1350 { \box_use_drop:N \l_@@_cell_box }
1351 {
1352 \bool_if:NTF \g_@@_not_empty_cell_bool
1353 \@@_node_for_cell:
1354 {
1355 \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1356 \@@_node_for_cell:
1357 { \box_use_drop:N \l_@@_cell_box }
1358 }
1359 }
1360 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1361 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1362 \bool_gset_false:N \g_@@_empty_cell_bool
1363 \bool_gset_false:N \g_@@_not_empty_cell_bool
1364 }

```


The following command will be nullified in our redefinition of `\multicolumn`.

```

1365 \cs_new_protected:Npn \@@_update_max_cell_width:
1366 {
1367   \dim_gset:Nn \g_@@_max_cell_width_dim
1368   { \dim_max:n \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1369 }

```

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`).

```

1370 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1371 {
1372   \@@_math_toggle:
1373   \hbox_set_end:
1374   \bool_if:NF \g_@@_rotate_bool
1375   {
1376     \hbox_set:Nn \l_@@_cell_box
1377     {
1378       \makebox [ \l_@@_col_width_dim ] [ s ]
1379       { \hbox_unpack_drop:N \l_@@_cell_box }
1380     }
1381   }
1382   \@@_cell_end_i:
1383 }

```

```

1384 \pgfset
1385 {
1386   nicematrix / cell-node /.style =
1387   {
1388     inner~sep = \c_zero_dim ,
1389     minimum~width = \c_zero_dim
1390   }
1391 }

```

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

```

1392 \cs_new_protected:Npn \@@_node_for_cell:
1393 {
1394   \pgfpicture
1395   \pgfsetbaseline \c_zero_dim
1396   \pgfrememberpicturepositiononpagetrue
1397   \pgfset { nicematrix / cell-node }
1398   \pgfnode
1399   { rectangle }
1400   { base }
1401   {

```

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

```

1402   \set@color
1403   \box_use_drop:N \l_@@_cell_box
1404 }
1405 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1406 { \l_@@_pgf_node_code_tl }
1407 \str_if_empty:NF \l_@@_name_str
1408 {
1409   \pgfnodealias
1410   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1411   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1412 }
1413 \endpgfpicture
1414 }

```

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.

```

1415 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1416 {
1417   \cs_new_protected:Npn \@@_patch_node_for_cell:
1418   {
1419     \hbox_set:Nn \l_@@_cell_box
1420     {
1421       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1422       \hbox_overlap_left:n
1423       {
1424         \pgfsys@markposition
1425         { \@@_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.

```

1426       #1
1427     }
1428     \box_use:N \l_@@_cell_box
1429     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1430     \hbox_overlap_left:n
1431     {
1432       \pgfsys@markposition
1433       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1434     }
1435   }
1436 }
1437 }
1438 }

```

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

```

1439 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1440 {
1441   \@@_patch_node_for_cell:n
1442   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1443 }
1444 { \@@_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).

```

1445 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1446 {
1447   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce

```

```

1448 { g_@@_ #2 _ lines _ t1 }
1449 {
1450   \use:c { @@ _ draw _ #2 : nnn }
1451   { \int_use:N \c@iRow }
1452   { \int_use:N \c@jCol }
1453   { \exp_not:n { #3 } }
1454 }
1455 }

1456 \cs_generate_variant:Nn \@@_array:n { o }
1457 \cs_new_protected:Npn \@@_array:n
1458 {
1459 %   \begin{macrocode}
1460   \dim_set:Nn \col@sep
1461     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1462   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1463     { \cs_set_nopar:Npn \@halignto { } }
1464     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1465 \tabarray

```

`\l_@@_baseline_t1` 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:eeTF` is fully expandable and we need something fully expandable here. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1466 [ \str_if_eq:eeTF \l_@@_baseline_t1 c c t ]
1467 }

```

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. However, since version 2.6a (version for the Tagging Project), `array` uses `\ar@ialign` instead of `\ialign`. In that case, of course, you do a saving of `\ar@ialign`.

```

1468 \bool_if:NTF \c_@@_tagging_array_bool
1469 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1470 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1471 \cs_new_protected:Npn \@@_create_row_node:
1472 {
1473   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1474   {
1475     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1476     \@@_create_row_node_i:
1477   }
1478 }

1479 \cs_new_protected:Npn \@@_create_row_node_i:
1480 {

```

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

```

1481 \hbox
1482 {
1483   \bool_if:NT \l_@@_code_before_bool
1484   {
1485     \vtop
1486     {
1487       \skip_vertical:N 0.5\arrayrulewidth
1488       \pgfsys@markposition
1489       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1490       \skip_vertical:N -0.5\arrayrulewidth
1491     }
1492   }
1493   \pgfpicture

```

```

1494 \pgfrememberpicturepositiononpagetrue
1495 \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1496 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1497 \str_if_empty:NF \l_@@_name_str
1498 {
1499 \pgfnodealias
1500 { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1501 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1502 }
1503 \endpgfpicture
1504 }
1505 }

```

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

```

1506 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1507 \cs_new_protected:Npn \@@_everycr_i:
1508 {
1509 \bool_if:NT \c_@@_testphase_table_bool
1510 {
1511 \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1512 \tbl_update_cell_data_for_next_row:
1513 }
1514 \int_gzero:N \c@jCol
1515 \bool_gset_false:N \g_@@_after_col_zero_bool
1516 \bool_if:NF \g_@@_row_of_col_done_bool
1517 {
1518 \@@_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).

```

1519 \clist_if_empty:NF \l_@@_hlines_clist
1520 {
1521 \tl_if_eq:NMF \l_@@_hlines_clist \c_@@_all_tl
1522 {
1523 \@@_clist_if_in:NeT
1524 \l_@@_hlines_clist
1525 { \int_eval:n { \c@iRow + 1 } }
1526 }
1527 {

```

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.

```

1528 \int_compare:nNnT \c@iRow > { -1 }
1529 {
1530 \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1531 { \hrule height \arrayrulewidth width \c_zero_dim }
1532 }
1533 }
1534 }
1535 }
1536 }

```

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

```

1537 \cs_set_protected:Npn \@@_renew_dots:
1538 {
1539 \cs_set_eq:NN \ldots \@@_Ldots
1540 \cs_set_eq:NN \cdots \@@_Cdots
1541 \cs_set_eq:NN \vdots \@@_Vdots
1542 \cs_set_eq:NN \ddots \@@_Ddots
1543 \cs_set_eq:NN \iddots \@@_Iddots

```

```

1544 \cs_set_eq:NN \dots \@@_Ldots
1545 \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1546 }
1547 \cs_new_protected:Npn \@@_test_color_inside:
1548 {
1549 \bool_if:NF \l_@@_color_inside_bool
1550 {

```

We will issue an error only during the first run.

```

1551 \bool_if:NF \g_@@_aux_found_bool
1552 { \@@_error:n { without~color~inside } }
1553 }
1554 }

```

```

1555 \cs_new_protected:Npn \@@_redefine_everycr:
1556 { \everycr { \@@_everycr: } }
1557 \hook_gput_code:nnn { begindocument } { . }
1558 {
1559 \IfPackageLoadedT { colortbl }
1560 {
1561 \cs_set_protected:Npn \@@_redefine_everycr:
1562 {
1563 \CT@everycr
1564 {
1565 \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1566 \@@_everycr:
1567 }
1568 }
1569 }
1570 }

```

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.

```

1571 \hook_gput_code:nnn { begindocument } { . }
1572 {
1573 \IfPackageLoadedTF { booktabs }
1574 {
1575 \cs_new_protected:Npn \@@_patch_booktabs:
1576 { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1577 }
1578 { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1579 }

```

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`.

```

1580 \cs_new_protected:Npn \@@_some_initialization:
1581 {
1582 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }

```

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

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

```

1583 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1584 \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1585 \dim_gzero:N \g_@@_dp_ante_last_row_dim
1586 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1587 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1588 }

```

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

```

1589 \cs_new_protected:Npn \@@_pre_array_ii:
1590 {

```

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

```

1591 \int_gzero:N \g_@@_total_X_weight_int
1592 \@@_expand_clist:N \l_@@_hlines_clist
1593 \@@_expand_clist:N \l_@@_vlines_clist
1594 \@@_patch_booktabs:
1595 \box_clear_new:N \l_@@_cell_box
1596 \normalbaselines

```

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

```

1597 \bool_if:NT \l_@@_small_bool
1598 {
1599 \cs_set_nopar:Npn \arraystretch { 0.47 }
1600 \dim_set:Nn \arraycolsep { 1.45 pt }

```

By default, `\@@_tuning_key_small:` is no-op.

```

1601 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1602 }

```

```

1603 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1604 {
1605 \tl_put_right:Nn \@@_begin_of_row:
1606 {
1607 \pgfsys@markposition
1608 { \@@_env: - row - \int_use:N \c@iRow - base }
1609 }
1610 }

```

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`.

```

1611 \bool_if:NTF \c_@@_tagging_array_bool
1612 {
1613 \cs_set_nopar:Npn \ar@ialign
1614 {
1615 \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1616 \@@_redefine_everycr:
1617 \dim_zero:N \tabskip
1618 \@@_some_initialization:

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With this programming, we will have, in the cells of the array, a clean version of `\ar@ialign`.

```

1619 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1620 \halign
1621 }
1622 }

```

The following part will be deleted when we will delete the boolean `\c_@@_tagging_array_bool` (when we consider the version 2.6a of array is required).

```

1623 {
1624   \cs_set_nopar:Npn \ialign
1625   {
1626     \@@_redefine_everycr:
1627     \dim_zero:N \tabskip
1628     \@@_some_initialization:
1629     \cs_set_eq:NN \ialign \@@_old_ialign:
1630     \halign
1631   }
1632 }

```

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).

```

1633 \cs_set_eq:NN \@@_old_ldots \ldots
1634 \cs_set_eq:NN \@@_old_cdots \cdots
1635 \cs_set_eq:NN \@@_old_vdots \vdots
1636 \cs_set_eq:NN \@@_old_ddots \ddots
1637 \cs_set_eq:NN \@@_old_iddots \iddots
1638 \bool_if:NTF \l_@@_standard_cline_bool
1639 { \cs_set_eq:NN \cline \@@_standard_cline }
1640 { \cs_set_eq:NN \cline \@@_cline }
1641 \cs_set_eq:NN \Ldots \@@_Ldots
1642 \cs_set_eq:NN \Cdots \@@_Cdots
1643 \cs_set_eq:NN \Vdots \@@_Vdots
1644 \cs_set_eq:NN \Ddots \@@_Ddots
1645 \cs_set_eq:NN \Iddots \@@_Iddots
1646 \cs_set_eq:NN \Hline \@@_Hline:
1647 \cs_set_eq:NN \Hspace \@@_Hspace:
1648 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1649 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1650 \cs_set_eq:NN \Block \@@_Block:
1651 \cs_set_eq:NN \rotate \@@_rotate:
1652 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1653 \cs_set_eq:NN \dotfill \@@_dotfill:
1654 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1655 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1656 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1657 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1658 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1659 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1660 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1661 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1662 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1663 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1664 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1665 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1666 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1667 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1668 \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. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```

1669 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1670 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1671 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1672 \@@_revert_colortbl:

```

If there is one or several commands `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the `aux` file at a previous run. We use that information to start counting the tabular

notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

1673 \tl_if_exist:NT \l_@@_note_in_caption_tl
1674 {
1675   \tl_if_empty:NF \l_@@_note_in_caption_tl
1676   {
1677     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1678     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1679   }
1680 }

```

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).

```

1681 \seq_gclear:N \g_@@_multicolumn_cells_seq
1682 \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).

```

1683 \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 of rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1684 \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.

```

1685 \int_gzero_new:N \g_@@_col_total_int
1686 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1687 \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.

```

1688 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1689 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1690 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1691 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1692 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1693 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1694 \tl_gclear:N \g_nicematrix_code_before_tl
1695 \tl_gclear:N \g_@@_pre_code_before_tl
1696 }

```

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

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

```

1697 \cs_new_protected:Npn \@@_pre_array:
1698 {
1699   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1700   \int_gzero_new:N \c@iRow
1701   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1702   \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).

```

1703 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1704 {
1705   \bool_set_true:N \l_@@_last_row_without_value_bool
1706   \bool_if:NT \g_@@_aux_found_bool
1707     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1708 }
1709 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1710 {
1711   \bool_if:NT \g_@@_aux_found_bool
1712     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1713 }

```

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”.

```

1714 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1715 {
1716   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1717     {
1718       \dim_gset:Nn \g_@@_ht_last_row_dim
1719         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1720       \dim_gset:Nn \g_@@_dp_last_row_dim
1721         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1722     }
1723 }

1724 \seq_gclear:N \g_@@_cols_vlism_seq
1725 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1726 \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.

```

1727 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1728 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1729 \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.

```

1730 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1731 \@@_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.

```

1732 \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]`).

```

1733 \dim_zero_new:N \l_@@_left_delim_dim
1734 \dim_zero_new:N \l_@@_right_delim_dim
1735 \bool_if:NTF \g_@@_delims_bool
1736 {

```

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

```

1737 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1738 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1739 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1740 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1741 }
1742 {
1743 \dim_gset:Nn \l_@@_left_delim_dim
1744 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1745 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1746 }

```

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).

```

1747 \hbox_set:Nw \l_@@_the_array_box
1748 \bool_if:NT \c_@@_testphase_table_bool
1749 { \UseTaggingSocket { tbl / hmode / begin } }
1750 \skip_horizontal:N \l_@@_left_margin_dim
1751 \skip_horizontal:N \l_@@_extra_left_margin_dim
1752 \c_math_toggle_token
1753 \bool_if:NTF \l_@@_light_syntax_bool
1754 { \use:c { @@-light-syntax } }
1755 { \use:c { @@-normal-syntax } }
1756 }

```

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

```

1757 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1758 {
1759 \tl_set:Nn \l_tmpa_tl { #1 }
1760 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1761 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1762 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1763 \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...

```

1764 \@@_pre_array:
1765 }

```

10 The `\CodeBefore`

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

```

1766 \cs_new_protected:Npn \@@_pre_code_before:
1767 {

```

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).

```

1768 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1769 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1770 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1771 \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.

```

1772 \pgfsys@markposition { \@@_env: - position }
1773 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1774 \pgfpicture
1775 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1776 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1777 {
1778   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1779   \pgfcoordinate { \@@_env: - row - ##1 }
1780   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1781 }

```

Now, the recreation of the `col` nodes.

```

1782 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1783 {
1784   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1785   \pgfcoordinate { \@@_env: - col - ##1 }
1786   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1787 }

```

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

```

1788 \@@_create_diag_nodes:

```

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

```

1789 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1790 \endpgfpicture

```

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

```

1791 \@@_create_blocks_nodes:
1792 \IfPackageLoadedT { tikz }
1793 {
1794   \tikzset
1795   {
1796     every-picture / .style =
1797     { overlay , name~prefix = \@@_env: - }
1798   }
1799 }
1800 \cs_set_eq:NN \cellcolor \@@_cellcolor
1801 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1802 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1803 \cs_set_eq:NN \rowcolor \@@_rowcolor
1804 \cs_set_eq:NN \rowcolors \@@_rowcolors
1805 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1806 \cs_set_eq:NN \arraycolor \@@_arraycolor
1807 \cs_set_eq:NN \columncolor \@@_columncolor
1808 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1809 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1810 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1811 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1812 }

```

```

1813 \cs_new_protected:Npn \@@_exec_code_before:
1814 {
1815   \seq_gclear_new:N \g_@@_colors_seq

```

The sequence `\g_@@_colors_seq` will always contain as first element the special color `nocolor`: when that color is used, no color will be applied in the corresponding cells by the other coloring commands of `nicematrix`.

```

1816   \@@_add_to_colors_seq:nn { { nocolor } } { }
1817   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1818   \group_begin:

```

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

```

1819   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token

```

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

```

1820   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1821   { \@@_rescan_for_spanish:N \l_@@_code_before_tl }

```

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`.

```

1822   \exp_last_unbraced:No \@@_CodeBefore_keys:
1823   \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.

```

1824   \@@_actually_color:
1825   \l_@@_code_before_tl
1826   \q_stop
1827   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1828   \group_end:
1829   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1830   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1831 }

```

```

1832 \keys_define:nn { nicematrix / CodeBefore }
1833 {
1834   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1835   create-cell-nodes .default:n = true ,
1836   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1837   sub-matrix .value_required:n = true ,
1838   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1839   delimiters / color .value_required:n = true ,
1840   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1841 }

1842 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1843 {
1844   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1845   \@@_CodeBefore:w
1846 }

```

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

```

1847 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1848 {

```

```

1849 \bool_if:NT \g_@@_aux_found_bool
1850 {
1851   \@@_pre_code_before:
1852   #1
1853 }
1854 }

```

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.

```

1855 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1856 {
1857   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1858   {
1859     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1860     \pgfcoordinate { \@@_env: - row - ##1 - base }
1861     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1862     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1863     {
1864       \cs_if_exist:cT
1865       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1866       {
1867         \pgfsys@getposition
1868         { \@@_env: - ##1 - #####1 - NW }
1869         \@@_node_position:
1870         \pgfsys@getposition
1871         { \@@_env: - ##1 - #####1 - SE }
1872         \@@_node_position_i:
1873         \@@_pgf_rect_node:nnn
1874         { \@@_env: - ##1 - #####1 }
1875         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1876         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1877       }
1878     }
1879   }
1880   \int_step_inline:nn \c@iRow
1881   {
1882     \pgfnodealias
1883     { \@@_env: - ##1 - last }
1884     { \@@_env: - ##1 - \int_use:N \c@jCol }
1885   }
1886   \int_step_inline:nn \c@jCol
1887   {
1888     \pgfnodealias
1889     { \@@_env: - last - ##1 }
1890     { \@@_env: - \int_use:N \c@iRow - ##1 }
1891   }
1892   \@@_create_extra_nodes:
1893 }

1894 \cs_new_protected:Npn \@@_create_blocks_nodes:
1895 {
1896   \pgfpicture
1897   \pgf@relevantforpicturesizefalse
1898   \pgfrememberpicturepositiononpagetrue
1899   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1900   { \@@_create_one_block_node:nnnn ##1 }
1901   \endpgfpicture
1902 }

```

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

```

1903 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1904 {
1905   \tl_if_empty:nF { #5 }
1906   {
1907     \@@_qpoint:n { col - #2 }
1908     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1909     \@@_qpoint:n { #1 }
1910     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1911     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1912     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1913     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1914     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1915     \@@_pgf_rect_node:nnnnn
1916     { \@@_env: - #5 }
1917     { \dim_use:N \l_tmpa_dim }
1918     { \dim_use:N \l_tmpb_dim }
1919     { \dim_use:N \l_@@_tmpc_dim }
1920     { \dim_use:N \l_@@_tmpd_dim }
1921   }
1922 }

1923 \cs_new_protected:Npn \@@_patch_for_revtext:
1924 {
1925   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1926   \cs_set_eq:NN \insert@column \insert@column@array
1927   \cs_set_eq:NN \@classx \@classx@array
1928   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1929   \cs_set_eq:NN \@arraycr \@arraycr@array
1930   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1931   \cs_set_eq:NN \array \array@array
1932   \cs_set_eq:NN \@array \@array@array
1933   \cs_set_eq:NN \@tabular \@tabular@array
1934   \cs_set_eq:NN \@mkpream \@mkpream@array
1935   \cs_set_eq:NN \endarray \endarray@array
1936   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1937   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1938 }

```

11 The environment `{NiceArrayWithDelims}`

```

1939 \NewDocumentEnvironment { NiceArrayWithDelims }
1940 { m m O { } m ! O { } t \CodeBefore }
1941 {
1942   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1943   \@@_provide_pgfsyspdfmark:
1944   \bool_if:NT \g_@@_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.

```

1945   \bgroup

1946   \tl_gset:Nn \g_@@_left_delim_tl { #1 }

```

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).

```

1947 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1948 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1949 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1950 \int_gzero:N \g_@@_block_box_int
1951 \dim_zero:N \g_@@_width_last_col_dim
1952 \dim_zero:N \g_@@_width_first_col_dim
1953 \bool_gset_false:N \g_@@_row_of_col_done_bool
1954 \str_if_empty:NT \g_@@_name_env_str
1955 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1956 \bool_if:NTF \l_@@_tabular_bool
1957 \mode_leave_vertical:
1958 \@@_test_if_math_mode:
1959 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1960 \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.

```

1961 \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.

```

1962 \cs_if_exist:NT \tikz@library@external@loaded
1963 {
1964   \tikzexternaldisable
1965   \cs_if_exist:NT \ifstandalone
1966     { \tikzset { external / optimize = false } }
1967 }

```

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

```

1968 \int_gincr:N \g_@@_env_int
1969 \bool_if:NF \l_@@_block_auto_columns_width_bool
1970 { \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`).

```

1971 \seq_gclear:N \g_@@_blocks_seq
1972 \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` and the `\multicolumn`.

```

1973 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1974 \seq_gclear:N \g_@@_pos_of_xdots_seq
1975 \tl_gclear_new:N \g_@@_code_before_tl
1976 \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.

```

1977 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1978 {
1979   \bool_gset_true:N \g_@@_aux_found_bool
1980   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1981 }
1982 { \bool_gset_false:N \g_@@_aux_found_bool }

```

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.

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1983 \tl_gclear:N \g_@@_aux_tl
1984 \tl_if_empty:NF \g_@@_code_before_tl
1985 {
1986   \bool_set_true:N \l_@@_code_before_bool
1987   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1988 }
1989 \tl_if_empty:NF \g_@@_pre_code_before_tl
1990 { \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`.

```

1991 \bool_if:NTF \g_@@_delims_bool
1992 { \keys_set:nn { nicematrix / pNiceArray } }
1993 { \keys_set:nn { nicematrix / NiceArray } }
1994 { #3 , #5 }

```

```

1995 \@@_set_CT@arc@:o \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:`.

```

1996 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1997 }

```

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

```

1998 {
1999   \bool_if:NTF \l_@@_light_syntax_bool
2000   { \use:c { end @@-light-syntax } }
2001   { \use:c { end @@-normal-syntax } }
2002   \c_math_toggle_token
2003   \skip_horizontal:N \l_@@_right_margin_dim
2004   \skip_horizontal:N \l_@@_extra_right_margin_dim
2005
2006   % awful workaround
2007   \int_compare:nNtT \g_@@_col_total_int = \c_one_int
2008   {
2009     \dim_compare:nNtT \l_@@_columns_width_dim > \c_zero_dim
2010     {
2011       \skip_horizontal:N - \l_@@_columns_width_dim
2012       \bool_if:NTF \l_@@_tabular_bool
2013       { \skip_horizontal:n { - 2 \tabcolsep } }
2014       { \skip_horizontal:n { - 2 \arraycolsep } }
2015     }
2016   }
2017   \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.

```

2018 \bool_if:NT \l_@@_width_used_bool
2019 {
2020   \int_if_zero:nT \g_@@_total_X_weight_int
2021   { \@@_error_or_warning:n { width-without-X-columns } }
2022 }

```

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`.

```

2023 \int_compare:nNtT \g_@@_total_X_weight_int > \c_zero_int

```



```

2024 {
2025   \tl_gput_right:N\g_@@_aux_tl
2026   {
2027     \bool_set_true:N \l_@@_X_columns_aux_bool
2028     \dim_set:Nn \l_@@_X_columns_dim
2029     {
2030       \dim_compare:nNnTF
2031       {
2032         \dim_abs:n
2033         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2034       }
2035       <
2036       { 0.001 pt }
2037       { \dim_use:N \l_@@_X_columns_dim }
2038       {
2039         \dim_eval:n
2040         {
2041           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2042           / \int_use:N \g_@@_total_X_weight_int
2043           + \l_@@_X_columns_dim
2044         }
2045       }
2046     }
2047   }
2048 }

```

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

```

2049 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2050 {
2051   \bool_if:NF \l_@@_last_row_without_value_bool
2052   {
2053     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2054     {
2055       \@@_error:n { Wrong~last~row }
2056       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2057     }
2058   }
2059 }

```

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

```

2060 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2061 \bool_if:NTF \g_@@_last_col_found_bool
2062 { \int_gdecr:N \c@jCol }
2063 {
2064   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2065   { \@@_error:n { last~col~not~used } }
2066 }

```

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

```

2067 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2068 \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. 91).

```

2069 \int_if_zero:nT \l_@@_first_col_int
2070 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

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

```

2071 \bool_if:nTF { ! \g_@@_delims_bool }
2072 {
2073   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2074   \@@_use_arraybox_with_notes_c:
2075   {
2076     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2077     \@@_use_arraybox_with_notes_b:
2078     \@@_use_arraybox_with_notes:
2079   }
2080 }

```

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

```

2081 {
2082   \int_if_zero:nTF \l_@@_first_row_int
2083   {
2084     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2085     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2086   }
2087   { \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

```

2088   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2089   {
2090     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2091     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2092   }
2093   { \dim_zero:N \l_tmpb_dim }
2094   \hbox_set:Nn \l_tmpa_box
2095   {
2096     \c_math_toggle_token
2097     \@@_color:o \l_@@_delimiters_color_tl
2098     \exp_after:wN \left \g_@@_left_delim_tl
2099     \vcenter
2100     {

```

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.

```

2101       \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2102       \hbox
2103       {
2104         \bool_if:NNTF \l_@@_tabular_bool
2105         { \skip_horizontal:N -\tabcolsep }
2106         { \skip_horizontal:N -\arraycolsep }
2107         \@@_use_arraybox_with_notes_c:
2108         \bool_if:NNTF \l_@@_tabular_bool
2109         { \skip_horizontal:N -\tabcolsep }
2110         { \skip_horizontal:N -\arraycolsep }
2111       }

```

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

```

2112       \skip_vertical:N -\l_tmpb_dim
2113       \skip_vertical:N \arrayrulewidth
2114     }
2115     \exp_after:wN \right \g_@@_right_delim_tl
2116     \c_math_toggle_token
2117   }

```

Now, the box `\l_tmpa_box` is created with the correct delimiters.

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

We will put the box in the TeX flow. However, we have a small work to do when the option `delimiters/max-width` is used.

```

2118     \bool_if:NTF \l_@@_delimiters_max_width_bool
2119     {
2120         \@@_put_box_in_flow_bis:nn
2121         \g_@@_left_delim_tl
2122         \g_@@_right_delim_tl
2123     }
2124     \@@_put_box_in_flow:
2125 }
```

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. 92).

```

2126     \bool_if:NT \g_@@_last_col_found_bool
2127     { \skip_horizontal:N \g_@@_width_last_col_dim }
2128     \bool_if:NT \l_@@_preamble_bool
2129     {
2130         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2131         { \@@_warning_gredirect_none:n { columns-not-used } }
2132     }
2133     \@@_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.

```

2134     \egroup
```

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

```

2135     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2136     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2137     \iow_now:Ne \@mainaux
2138     {
2139         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2140         { \exp_not:o \g_@@_aux_tl }
2141     }
2142     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2143     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2144 }
```

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

12 We construct the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble which will be given to `{array}` (of the package `array`).

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

```

2145     \cs_new_protected:Npn \@@_transform_preamble:
2146     {
2147         \@@_transform_preamble_i:
2148         \@@_transform_preamble_ii:
2149     }

2150     \cs_new_protected:Npn \@@_transform_preamble_i:
2151     {
2152         \int_gzero:N \c@jCol
```

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`).

```

2153     \seq_gclear:N \g_@@_cols_vlism_seq
```

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

```
2154 \bool_gset_false:N \g_tmpb_bool
```

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

```
2155 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2156 \int_zero:N \l_tmpa_int
2157 \tl_gclear:N \g_@@_array_preamble_tl
2158 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2159 {
2160   \tl_gset:Nn \g_@@_array_preamble_tl
2161     { ! { \skip_horizontal:N \arrayrulewidth } }
2162 }
2163 {
2164   \@@_clist_if_in:NnT \l_@@_vlines_clist 1
2165   {
2166     \tl_gset:Nn \g_@@_array_preamble_tl
2167       { ! { \skip_horizontal:N \arrayrulewidth } }
2168   }
2169 }
```

Now, we actually make the preamble (which will be given to `{array}`). It will be stored in `\g_@@_array_preamble_tl`.

```
2170 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2171 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2172 \@@_replace_columncolor:
2173 }
```

```
2174 \hook_gput_code:nnn { begindocument } { . }
2175 {
2176   \IfPackageLoadedTF { colortbl }
2177   {
```

When `colortbl` is used, we have to catch the tokens `\columncolor` in the preamble because, otherwise, `colortbl` will catch them and the colored panels won't be drawn by `nicematrix` but by `colortbl` (with an output which is not perfect).

```
2178   \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2179   \cs_new_protected:Npn \@@_replace_columncolor:
2180   {
2181     \regex_replace_all:NnN
2182       \c_@@_columncolor_regex
2183       { \c { @@_columncolor_preamble } }
2184       \g_@@_array_preamble_tl
2185   }
2186 }
2187 {
2188   \cs_new_protected:Npn \@@_replace_columncolor:
2189   { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2190 }
2191 }
```

```
2192 \cs_new_protected:Npn \@@_transform_preamble_ii:
2193 {
```

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

```
2194 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2195 {
2196   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
```

```

2197         { \bool_gset_true:N \g_@@_delims_bool }
2198     }
2199     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2200     \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).

```

2201     \int_if_zero:nTF \l_@@_first_col_int
2202     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2203     {
2204         \bool_if:NF \g_@@_delims_bool
2205         {
2206             \bool_if:NF \l_@@_tabular_bool
2207             {
2208                 \clist_if_empty:NT \l_@@_vlines_clist
2209                 {
2210                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2211                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2212                 }
2213             }
2214         }
2215     }
2216     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2217     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2218     {
2219         \bool_if:NF \g_@@_delims_bool
2220         {
2221             \bool_if:NF \l_@@_tabular_bool
2222             {
2223                 \clist_if_empty:NT \l_@@_vlines_clist
2224                 {
2225                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2226                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2227                 }
2228             }
2229         }
2230     }

```

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).

```

2231     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2232     {
2233         \tl_gput_right:Nn \g_@@_array_preamble_tl
2234         { > { \@@_error_too_much_cols: } 1 }
2235     }
2236 }

```

The preamble provided by the final user will be read by a finite automata. The following function \@@_rec_preamble:n will read that preamble (usually letter by letter) in a recursive way (hence the name of that function). in the preamble.

```

2237 \cs_new_protected:Npn \@@_rec_preamble:n #1
2238 {

```

For the majority of the letters, we will trigger the corresponding action by calling directly a function in the main hashtable of TeX (thanks to the mechanism \csname...\endcsname. Be careful: all these functions take in as first argument the letter (or token) itself.¹⁰

¹⁰We do that because it’s an easy way to insert the letter at some places in the code that we will add to \g_@@_array_preamble_tl.

```

2239 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2240 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2241 {

```

Now, the columns defined by \newcolumntype of array.

```

2242 \cs_if_exist:cTF { NC @ find @ #1 }
2243 {
2244 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2245 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2246 }
2247 {

```

Remember that #1 is a token.

```

2248 \token_if_eq_meaning:NNTF #1 S
2249 { \@@_fatal:n { unknown~column~type~S } }
2250 { \@@_fatal:nn { unknown~column~type } { #1 } }
2251 }
2252 }
2253 }

```

For c, l and r

```

2254 \cs_new:Npn \@@_c #1
2255 {
2256 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2257 \tl_gclear:N \g_@@_pre_cell_tl
2258 \tl_gput_right:Nn \g_@@_array_preamble_tl
2259 { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2260 \int_gincr:N \c@jCol
2261 \@@_rec_preamble_after_col:n
2262 }

2263 \cs_new:Npn \@@_l #1
2264 {
2265 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2266 \tl_gclear:N \g_@@_pre_cell_tl
2267 \tl_gput_right:Nn \g_@@_array_preamble_tl
2268 {
2269 > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2270 l
2271 < \@@_cell_end:
2272 }
2273 \int_gincr:N \c@jCol
2274 \@@_rec_preamble_after_col:n
2275 }

2276 \cs_new:Npn \@@_r #1
2277 {
2278 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2279 \tl_gclear:N \g_@@_pre_cell_tl
2280 \tl_gput_right:Nn \g_@@_array_preamble_tl
2281 {
2282 > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2283 r
2284 < \@@_cell_end:
2285 }
2286 \int_gincr:N \c@jCol
2287 \@@_rec_preamble_after_col:n
2288 }

```

For ! and @

```

2289 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2290 {
2291 \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }

```

```

2292 \@@_rec_preamble:n
2293 }
2294 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2295 \cs_new:cpn { @@ _ | } #1
2296 {
\l_tmpa_int is the number of successive occurrences of |
2297 \int_incr:N \l_tmpa_int
2298 \@@_make_preamble_i_i:n
2299 }
2300 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2301 {
2302 \token_if_eq_meaning:NNTF #1 |
2303 { \use:c { @@ _ | } | }
2304 { \@@_make_preamble_i_ii:nn { } #1 }
2305 }
2306 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2307 {
2308 \token_if_eq_meaning:NNTF #2 [
2309 { \@@_make_preamble_i_iii:nw { #1 } [ }
2310 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2311 ]
2312 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2313 { \@@_make_preamble_i_iii:nn { #1 , #2 } }
2314 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2315 {
2316 \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2317 \tl_gput_right:Ne \g_@@_array_preamble_tl
2318 {

```

Here, the command \dim_eval:n is mandatory.

```

2319 \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2320 }
2321 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2322 {
2323 \@@_vline:n
2324 {
2325 position = \int_eval:n { \c@jCol + 1 } ,
2326 multiplicity = \int_use:N \l_tmpa_int ,
2327 total-width = \dim_use:N \l_@@_rule_width_dim ,
2328 #2
2329 }

```

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.

```

2330 }
2331 \int_zero:N \l_tmpa_int
2332 \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2333 \@@_rec_preamble:n #1
2334 }
2335 \cs_new:cpn { @@ _ > } #1 #2
2336 {
2337 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2338 \@@_rec_preamble:n
2339 }
2340 \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.

```

2341 \keys_define:nn { nicematrix / p-column }
2342 {
2343   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2344   r .value_forbidden:n = true ,
2345   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2346   c .value_forbidden:n = true ,
2347   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2348   l .value_forbidden:n = true ,
2349   R .code:n =
2350     \IfPackageLoadedTF { ragged2e }
2351     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2352     {
2353       \@@_error_or_warning:n { ragged2e~not~loaded }
2354       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2355     } ,
2356   R .value_forbidden:n = true ,
2357   L .code:n =
2358     \IfPackageLoadedTF { ragged2e }
2359     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2360     {
2361       \@@_error_or_warning:n { ragged2e~not~loaded }
2362       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2363     } ,
2364   L .value_forbidden:n = true ,
2365   C .code:n =
2366     \IfPackageLoadedTF { ragged2e }
2367     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2368     {
2369       \@@_error_or_warning:n { ragged2e~not~loaded }
2370       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2371     } ,
2372   C .value_forbidden:n = true ,
2373   S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2374   S .value_forbidden:n = true ,
2375   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2376   p .value_forbidden:n = true ,
2377   t .meta:n = p ,
2378   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2379   m .value_forbidden:n = true ,
2380   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2381   b .value_forbidden:n = true ,
2382 }

```

For `p` but also `b` and `m`.

```

2383 \cs_new:Npn \@@_p #1
2384 {
2385   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2386   \@@_make_preamble_ii_i:n
2387 }
2388 \cs_set_eq:NN \@@_b \@@_p
2389 \cs_set_eq:NN \@@_m \@@_p
2390 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2391 {
2392   \token_if_eq_meaning:NNTF #1 [
2393     { \@@_make_preamble_ii_ii:w [ ] }
2394     { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2395   }
2396 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2397 { \@@_make_preamble_ii_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.

```
2398 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2399 {
```

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).

```
2400   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2401   \@@_keys_p_column:n { #1 }
2402   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2403 }

2404 \cs_new_protected:Npn \@@_keys_p_column:n #1
2405 { \keys_set_known:nnN { nicematrix / p-column } { #1 } \l_tmpa_tl }
```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`. The third is some code added at the beginning of the cell.

```
2406 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2407 {
2408   \use:e
2409   {
2410     \@@_make_preamble_ii_v:nnnnnnnn
2411     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2412     { \dim_eval:n { #1 } }
2413     {
```

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

```
2414       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2415       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2416       {
```

Here, we use `\cs_set_nopar:Npn` instead of `\tl_set:Nn` for efficiency only.

```
2417         \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2418         { \str_lowercase:o \l_@@_hpos_col_str }
2419       }
2420   \str_case:on \l_@@_hpos_col_str
2421   {
2422     c { \exp_not:N \centering }
2423     l { \exp_not:N \raggedright }
2424     r { \exp_not:N \raggedleft }
2425     C { \exp_not:N \Centering }
2426     L { \exp_not:N \RaggedRight }
2427     R { \exp_not:N \RaggedLeft }
2428   }
2429   #3
2430 }
2431 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2432 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2433 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2434 { #2 }
2435 {
2436   \str_case:onF \l_@@_hpos_col_str
2437   {
2438     { j } { c }
2439     { si } { c }
2440   }
```

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

```
2441     { \str_lowercase:o \l_@@_hpos_col_str }
2442   }
2443 }
```

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

```

2444 \int_gincr:N \c@jCol
2445 \@@_rec_preamble_after_col:n
2446 }

```

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

```

2447 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2448 {
2449   \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2450   {
2451     \tl_gput_right:Nn \g_@@_array_preamble_tl
2452     { > { \@@_test_if_empty_for_S: } }
2453   }
2454   {
2455     \tl_gput_right:Nn \g_@@_array_preamble_tl
2456     { > { \@@_test_if_empty: } }
2457   }
2458   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2459   \tl_gclear:N \g_@@_pre_cell_tl
2460   \tl_gput_right:Nn \g_@@_array_preamble_tl
2461   {
2462     > {

```

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.

```

2463 \dim_set:Nn \l_@@_col_width_dim { #2 }
2464 \bool_if:NT \c_@@_testphase_table_bool
2465 { \tag_struct_begin:n { tag = Div } }
2466 \@@_cell_begin:w

```

We use the form \minipage–\endminipage (\varwidth–\endvarwidth) for compatibility with collcell (2023-10-31).

```

2467 \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from array.sty.

```

2468 \everypar
2469 {
2470   \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2471   \everypar { }
2472 }
2473 \bool_if:NT \c_@@_testphase_table_bool \tagpdfparaOn

```

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

```

2474 #3

```

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

```

2475 \g_@@_row_style_tl
2476 \arraybackslash
2477 #5
2478 }
2479 #8

```

```

2480 < {
2481     #6

```

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

```

2482 \@@finalstrut \@arstrutbox
2483 \use:c { end #7 }

```

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

```

2484     #4
2485     \@@_cell_end:
2486     \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2487 }
2488 }
2489 }

```

```

2490 \str_new:N \c_@@_ignorespaces_str
2491 \str_set:Ne \c_@@_ignorespaces_str { \ignorespaces }
2492 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }
2493 \cs_new_protected:Npn \@@_test_if_empty:
2494 { \peek_after:Nw \@@_test_if_empty_i: }
2495 \cs_new_protected:Npn \@@_test_if_empty_i:
2496 {
2497     \str_set:Ne \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2498     \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2499     { \@@_test_if_empty:w }
2500 }
2501 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2502 { \peek_after:Nw \@@_test_if_empty_ii: }

```

```

2503 \cs_new_protected:Npn \@@_nullify_cell:
2504 {
2505     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2506     {
2507         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2508         \skip_horizontal:N \l_@@_col_width_dim
2509     }
2510 }

```

```

2511 \bool_if:NNT \c_@@_tagging_array_bool
2512 {
2513     \cs_new_protected:Npn \@@_test_if_empty_ii:
2514     { \peek_meaning:NT \textonly@unskip \@@_nullify_cell: }
2515 }

```

In the old version of `array`, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstances, for example when `\collectcell` of `colcell` is used, the cell does not begin with `\ignorespaces`. In that case, we consider as not empty... First, we test if the next token is `\ignorespaces` and it's not very easy...

```

2516 {
2517     \cs_new_protected:Npn \@@_test_if_empty_ii:
2518     { \peek_meaning:NT \unskip \@@_nullify_cell: }
2519 }
2520 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2521 {
2522     \peek_meaning:NT \__siunitx_table_skip:n
2523     {
2524         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2525         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2526     }
2527 }

```

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

```
2528 \cs_new_protected:Npn \@@_center_cell_box:
2529 {
```

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

```
2530 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2531 {
2532   \int_compare:nNnT
2533     { \box_ht:N \l_@@_cell_box }
2534     >
```

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).

```
2535     { \box_ht:N \strutbox }
2536     {
2537       \hbox_set:Nn \l_@@_cell_box
2538         {
2539           \box_move_down:nn
2540             {
2541               ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2542                 + \baselineskip ) / 2
2543             }
2544           { \box_use:N \l_@@_cell_box }
2545         }
2546       }
2547     }
2548   }
```

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

```
2549 \cs_new:Npn \@@_V #1 #2
2550 {
2551   \token_if_eq_meaning:NNTF #1 [
2552     { \@@_make_preamble_V_i:w [ ] }
2553     { \@@_make_preamble_V_i:w [ ] { #2 } }
2554   }
2555   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2556     { \@@_make_preamble_V_ii:nn { #1 } }
2557   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2558     {
2559       \str_set:Nn \l_@@_vpos_col_str { p }
2560       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2561       \@@_keys_p_column:n { #1 }
2562       \IfPackageLoadedTF { varwidth }
2563         { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2564         {
2565           \@@_error_or_warning:n { varwidth-not-loaded }
2566           \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2567         }
2568     }
```

For `w` and `W`

```
2569 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2570 \cs_new:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_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.

```

2571 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2572 {
2573   \token_if_eq_meaning:NNTF #3 s
2574   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2575   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2576 }

```

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.

```

2577 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2578 {
2579   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2580   \tl_gclear:N \g_@@_pre_cell_tl
2581   \tl_gput_right:Nn \g_@@_array_preamble_tl
2582   {
2583     > {
2584       \dim_set:Nn \l_@@_col_width_dim { #2 }
2585       \@@_cell_begin:w
2586       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2587     }
2588     c
2589     < {
2590       \@@_cell_end_for_w_s:
2591       #1
2592       \@@_adjust_size_box:
2593       \box_use_drop:N \l_@@_cell_box
2594     }
2595   }
2596   \int_gincr:N \c_jCol
2597   \@@_rec_preamble_after_col:n
2598 }

```

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

```

2599 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2600 {
2601   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2602   \tl_gclear:N \g_@@_pre_cell_tl
2603   \tl_gput_right:Nn \g_@@_array_preamble_tl
2604   {
2605     > {

```

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.

```

2606       \dim_set:Nn \l_@@_col_width_dim { #4 }
2607       \hbox_set:Nw \l_@@_cell_box
2608       \@@_cell_begin:w
2609       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2610     }
2611     c
2612     < {
2613       \@@_cell_end:
2614       \hbox_set_end:
2615       #1
2616       \@@_adjust_size_box:
2617       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2618     }
2619   }

```

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

```

2620 \int_gincr:N \c@jCol
2621 \@@_rec_preamble_after_col:n
2622 }

2623 \cs_new_protected:Npn \@@_special_W:
2624 {
2625 \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2626 { \@@_warning:n { W~warning } }
2627 }

```

For S (of siunitx).

```

2628 \cs_new:Npn \@@_S #1 #2
2629 {
2630 \token_if_eq_meaning:NNTF #1 [
2631 { \@@_make_preamble_S:w [ ] }
2632 { \@@_make_preamble_S:w [ ] { #2 } }
2633 ]

2634 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2635 { \@@_make_preamble_S_i:n { #1 } }

2636 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2637 {
2638 \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2639 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2640 \tl_gclear:N \g_@@_pre_cell_tl
2641 \tl_gput_right:Nn \g_@@_array_preamble_tl
2642 {
2643 > {
2644 \@@_cell_begin:w
2645 \keys_set:nn { siunitx } { #1 }
2646 \siunitx_cell_begin:w
2647 }
2648 c
2649 < { \siunitx_cell_end: \@@_cell_end: }
2650 }

```

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

```

2651 \int_gincr:N \c@jCol
2652 \@@_rec_preamble_after_col:n
2653 }

```

For (, [and \{.

```

2654 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2655 {
2656 \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.

```

2657 \int_if_zero:nTF \c@jCol
2658 {
2659 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2660 {

```

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

```

2661 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2662 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2663 \@@_rec_preamble:n #2
2664 }
2665 {
2666 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2667 \@@_make_preamble_iv:nn { #1 } { #2 }
2668 }

```

```

2669     }
2670     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2671   }
2672   \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2673   \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2674   \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2675   {
2676     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2677     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2678     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2679     {
2680       \@@_error:nn { delimiter~after~opening } { #2 }
2681       \@@_rec_preamble:n
2682     }
2683     { \@@_rec_preamble:n #2 }
2684   }

```

In fact, it would be possible to define `\left` and `\right` as no-op.

```

2685   \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

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}`).

```

2686   \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2687   {
2688     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2689     \tl_if_in:nnTF { ) ] \} } { #2 }
2690     { \@@_make_preamble_v:nnn #1 #2 }
2691     {
2692       \str_if_eq:nnTF { \@@_stop: } { #2 }
2693       {
2694         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2695         { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2696         {
2697           \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2698           \tl_gput_right:Ne \g_@@_pre_code_after_tl
2699           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2700           \@@_rec_preamble:n #2
2701         }
2702       }
2703       {
2704         \tl_if_in:nnT { ( [ \{ \left } { #2 }
2705         { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2706         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2707         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2708         \@@_rec_preamble:n #2
2709       }
2710     }
2711   }
2712   \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2713   \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2714   \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2715   {
2716     \str_if_eq:nnTF { \@@_stop: } { #3 }
2717     {
2718       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2719       {
2720         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2721         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2722         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2723         \tl_gset:Nn \g_@@_right_delim_tl { #2 }

```

```

2724     }
2725     {
2726         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2727         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2728         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2729         \@@_error:nn { double~closing~delimiter } { #2 }
2730     }
2731 }
2732 {
2733     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2734     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2735     \@@_error:nn { double~closing~delimiter } { #2 }
2736     \@@_rec_preamble:n #3
2737 }
2738 }

2739 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2740 { \use:c { @@ _ \token_to_str:N ) } }

```

After a specifier of column, we have to test whether there is one or several $\langle \{ \dots \}$ because, after those potential $\langle \{ \dots \}$, we have to insert $!\{\backslash\text{skip_horizontal:N } \dots\}$ when the key `vlines` is used. In fact, we have also to test whether there is, after the $\langle \{ \dots \}$, a $\@ \{ \dots \}$.

```

2741 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2742 {
2743     \token_if_eq_meaning:NNTF #1 <
2744     \@@_rec_preamble_after_col_i:n
2745     {
2746         \token_if_eq_meaning:NNTF #1 @
2747         \@@_rec_preamble_after_col_ii:n
2748         {
2749             \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2750             {
2751                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2752                 { ! { \skip_horizontal:N \arrayrulewidth } }
2753             }
2754             {
2755                 \@@_clist_if_in:NeT \l_@@_vlines_clist
2756                 { \int_eval:n { \c@jCol + 1 } }
2757                 {
2758                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2759                     { ! { \skip_horizontal:N \arrayrulewidth } }
2760                 }
2761             }
2762             \@@_rec_preamble:n { #1 }
2763         }
2764     }
2765 }

2766 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2767 {
2768     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2769     \@@_rec_preamble_after_col:n
2770 }

```

We have to catch a $\@ \{ \dots \}$ after a specifier of column because, if we have to draw a vertical rule, we have to add in that $\@ \{ \dots \}$ a $\backslash\text{hskip}$ corresponding to the width of the vertical rule.

```

2771 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2772 {
2773     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2774     {
2775         \tl_gput_right:Nn \g_@@_array_preamble_tl
2776         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2777     }

```



```

2778 {
2779   \@@_clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2780   {
2781     \tl_gput_right:Nn \g_@@_array_preamble_tl
2782     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2783   }
2784   { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2785 }
2786 \@@_rec_preamble:n
2787 }

2788 \cs_new:cpn { @@ _ * } #1 #2 #3
2789 {
2790   \tl_clear:N \l_tmpa_tl
2791   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2792   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2793 }

```

The token `\NC@find` is at the head of the definition of the columns type done by `\newcolumnntype`. We want that token to be no-op here.

```

2794 \cs_new:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }

```

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`.

```

2795 \cs_new:Npn \@@_X #1 #2
2796 {
2797   \token_if_eq_meaning:NNTF #2 [
2798     { \@@_make_preamble_X:w [ ] }
2799     { \@@_make_preamble_X:w [ ] #2 }
2800   }

2801   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2802   { \@@_make_preamble_X:i: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 `{ nicematrix / 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`).

```

2803 \keys_define:nn { nicematrix / X-column }
2804 { 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`.

```

2805 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2806 {

```

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`).

```

2807   \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`).

```

2808   \str_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`.

```

2809   \int_zero_new:N \l_@@_weight_int
2810   \int_set_eq:NN \l_@@_weight_int \c_one_int
2811   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in \l_tmpa_tl

```

2812 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2813 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2814 {
2815   \@@_error_or_warning:n { negative-weight }
2816   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2817 }
2818 \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).

```

2819 \bool_if:NTF \l_@@_X_columns_aux_bool
2820 {
2821   \@@_make_preamble_ii_iv:nnn
2822   { \l_@@_weight_int \l_@@_X_columns_dim }
2823   { minipage }
2824   { \@@_no_update_width: }
2825 }
2826 {
2827   \tl_gput_right:Nn \g_@@_array_preamble_tl
2828   {
2829     > {
2830       \@@_cell_begin:w
2831       \bool_set_true:N \l_@@_X_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.

```

2832 \NotEmpty

```

The following code will nullify the box of the cell.

```

2833 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2834 { \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.

```

2835 \begin { minipage } { 5 cm } \arraybackslash
2836 }
2837 c
2838 < {
2839   \end { minipage }
2840   \@@_cell_end:
2841 }
2842 }
2843 \int_gincr:N \c@jCol
2844 \@@_rec_preamble_after_col:n
2845 }
2846 }

2847 \cs_new_protected:Npn \@@_no_update_width:
2848 {
2849   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2850   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2851 }

```

For the letter set by the user with vlines-in-sub-matrix (vlism).

```

2852 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2853 {
2854   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2855   { \int_eval:n { \c@jCol + 1 } }
2856   \tl_gput_right:Ne \g_@@_array_preamble_tl
2857   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2858   \@@_rec_preamble:n
2859 }

```

The token `\@@_stop:` is a marker that we have inserted to mark the end of the preamble (as provided by the final user) that we have inserted in the TeX flow.

```
2860 \cs_set_eq:cN { @@ _ \token_to_str:N \@@_stop: } \use_none:n
```

The following lines try to catch some errors (when the final user has forgotten the preamble of its environment).

```
2861 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2862 { \@@_fatal:n { Preamble-forgotten } }
2863 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2864 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2865 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
```

13 The redefinition of `\multicolumn`

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

```
2866 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2867 {
```

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`.

```
2868 \multispan { #1 }
2869 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2870 \begingroup
2871 \bool_if:NT \c_@@_testphase_table_bool
2872 { \tbl_update_multicolumn_cell_data:n { #1 } }
2873 \cs_set_nopar:Npn \@addamp
2874 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }
```

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

```
2875 \tl_gclear:N \g_@@_preamble_tl
2876 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2877 \exp_args:No \mkpream \g_@@_preamble_tl
2878 \@addtopreamble \empty
2879 \endgroup
2880 \bool_if:NT \c_@@_testphase_table_bool
2881 { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2882 \int_compare:nNnT { #1 } > \c_one_int
2883 {
2884 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2885 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2886 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2887 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2888 {
2889 {
2890 \int_if_zero:nTF \c@jCol
2891 { \int_eval:n { \c@iRow + 1 } }
2892 { \int_use:N \c@iRow }
2893 }
2894 { \int_eval:n { \c@jCol + 1 } }
2895 {
2896 \int_if_zero:nTF \c@jCol
2897 { \int_eval:n { \c@iRow + 1 } }
```

```

2898         { \int_use:N \c@iRow }
2899     }
2900     { \int_eval:n { \c@jCol + #1 } }
2901     { } % for the name of the block
2902 }
2903 }

```

We want `\cellcolor` to be available in `\multicolumn` because `\cellcolor` of `colortbl` is available in `\multicolumn`.

```

2904 \RenewDocumentCommand \cellcolor { 0 { } m }
2905 {
2906     \@@_test_color_inside:
2907     \tl_gput_right:Nx \g_@@_pre_code_before_tl
2908     {
2909         \@@_rectanglecolor [ ##1 ]
2910         { \exp_not:n { ##2 } }
2911         { \int_use:N \c@iRow - \int_use:N \c@jCol }
2912         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2913     }
2914     \ignorespaces
2915 }

```

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

```

2916 \cs_set_nopar:Npn \@sharp { #3 }
2917 \@arstrut
2918 \@preamble
2919 \null

```

We add some lines.

```

2920 \int_gadd:Nn \c@jCol { #1 - 1 }
2921 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2922 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2923 \ignorespaces
2924 }

```

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`.

```

2925 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2926 {
2927     \str_case:nnF { #1 }
2928     {
2929         c { \@@_make_m_preamble_i:n #1 }
2930         l { \@@_make_m_preamble_i:n #1 }
2931         r { \@@_make_m_preamble_i:n #1 }
2932         > { \@@_make_m_preamble_ii:nn #1 }
2933         ! { \@@_make_m_preamble_ii:nn #1 }
2934         @ { \@@_make_m_preamble_ii:nn #1 }
2935         | { \@@_make_m_preamble_iii:n #1 }
2936         p { \@@_make_m_preamble_iv:nnn t #1 }
2937         m { \@@_make_m_preamble_iv:nnn c #1 }
2938         b { \@@_make_m_preamble_iv:nnn b #1 }
2939         w { \@@_make_m_preamble_v:nnnn { } #1 }
2940         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2941         \q_stop { }
2942     }
2943     {
2944         \cs_if_exist:cTF { NC @ find @ #1 }
2945         {
2946             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2947             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2948         }
2949         {

```

Remember that #1 is a token.

```

2950         \token_if_eq_meaning:NNTF #1 S
2951         { \@@_fatal:n { unknown~column~type~S } }
2952         { \@@_fatal:nn { unknown~column~type } { #1 } }
2953     }
2954 }
2955 }

```

For c, l and r

```

2956 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2957 {
2958     \tl_gput_right:Nn \g_@@_preamble_tl
2959     {
2960         > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2961         #1
2962         < \@@_cell_end:
2963     }

```

We test for the presence of a <.

```

2964     \@@_make_m_preamble_x:n
2965 }

```

For >, ! and @

```

2966 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2967 {
2968     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2969     \@@_make_m_preamble:n
2970 }

```

For l

```

2971 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2972 {
2973     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2974     \@@_make_m_preamble:n
2975 }

```

For p, m and b

```

2976 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2977 {
2978     \tl_gput_right:Nn \g_@@_preamble_tl
2979     {
2980         > {
2981             \@@_cell_begin:w
2982             \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2983             \mode_leave_vertical:
2984             \arraybackslash
2985             \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2986         }
2987         c
2988         < {
2989             \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2990             \end { minipage }
2991             \@@_cell_end:
2992         }
2993     }

```

We test for the presence of a <.

```

2994     \@@_make_m_preamble_x:n
2995 }

```

For w and W

```

2996 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2997 {

```

```

2998 \tl_gput_right:Nn \g_@@_preamble_tl
2999 {
3000   > {
3001     \dim_set:Nn \l_@@_col_width_dim { #4 }
3002     \hbox_set:Nw \l_@@_cell_box
3003     \@@_cell_begin:w
3004     \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
3005   }
3006   c
3007   < {
3008     \@@_cell_end:
3009     \hbox_set_end:
3010     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3011     #1
3012     \@@_adjust_size_box:
3013     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3014   }
3015 }

```

We test for the presence of a <.

```

3016 \@@_make_m_preamble_x:n
3017 }

```

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

```

3018 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3019 {
3020   \token_if_eq_meaning:NNTF #1 <
3021   \@@_make_m_preamble_ix:n
3022   { \@@_make_m_preamble:n { #1 } }
3023 }
3024 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3025 {
3026   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3027   \@@_make_m_preamble_x:n
3028 }

```

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).

```

3029 \cs_new_protected:Npn \@@_put_box_in_flow:
3030 {
3031   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3032   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3033   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
3034   { \box_use_drop:N \l_tmpa_box }
3035   \@@_put_box_in_flow_i:
3036 }

```

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).

```

3037 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3038 {
3039   \pgfpicture
3040   \@@_qpoint:n { row - 1 }
3041   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3042   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3043   \dim_gadd:Nn \g_tmpa_dim \pgf@y
3044   \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).

```

3045     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3046     {
3047         \int_set:Nn \l_tmpa_int
3048         {
3049             \str_range:Nnn
3050             \l_@@_baseline_tl
3051             6
3052             { \tl_count:o \l_@@_baseline_tl }
3053         }
3054         \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3055     }
3056     {
3057         \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3058         { \int_set_eq:NN \l_tmpa_int \c_one_int }
3059         {
3060             \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3061             { \int_set_eq:NN \l_tmpa_int \c_iRow }
3062             { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3063         }
3064         \bool_lazy_or:nnT
3065         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3066         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3067         {
3068             \@@_error:n { bad-value-for-baseline }
3069             \int_set_eq:NN \l_tmpa_int \c_one_int
3070         }
3071         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3072         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3073     }
3074     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3075     \endpgfpicture
3076     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3077     \box_use_drop:N \l_tmpa_box
3078 }

```

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).

```

3079 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3080 {

```

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.

```

3081     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3082     {
3083         \int_compare:nNnT \c_jCol > \c_one_int
3084         {
3085             \box_set_wd:Nn \l_@@_the_array_box
3086             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3087         }
3088     }

```

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

```

3089     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3090     \bool_if:NT \l_@@_caption_above_bool
3091     {

```

```

3092     \tl_if_empty:NF \l_@@_caption_tl
3093     {
3094         \bool_set_false:N \g_@@_caption_finished_bool
3095         \int_gzero:N \c@tabularnote
3096         \@@_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).

```

3097         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3098         {
3099             \tl_gput_right:Ne \g_@@_aux_tl
3100             {
3101                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3102                 { \int_use:N \g_@@_notes_caption_int }
3103             }
3104             \int_gzero:N \g_@@_notes_caption_int
3105         }
3106     }
3107 }

```

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

```

3108     \hbox
3109     {
3110         \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.

```

3111     \@@_create_extra_nodes:
3112     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3113 }

```

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

```

3114     \bool_lazy_any:nT
3115     {
3116         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3117         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3118         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3119     }
3120     \@@_insert_tabularnotes:
3121     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3122     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3123     \end { minipage }
3124 }

```

```

3125 \cs_new_protected:Npn \@@_insert_caption:
3126 {
3127     \tl_if_empty:NF \l_@@_caption_tl
3128     {
3129         \cs_if_exist:NTF \c@type
3130         { \@@_insert_caption_i: }
3131         { \@@_error:n { caption~outside~float } }
3132     }
3133 }

```

```

3134 \cs_new_protected:Npn \@@_insert_caption_i:
3135 {
3136     \group_begin:

```


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

```
3137 \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.

```
3138 \IfPackageLoadedT { floatrow }
3139 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3140 \tl_if_empty:NTF \l_@@_short_caption_tl
3141 { \caption }
3142 { \caption [ \l_@@_short_caption_tl ] }
3143 { \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.

```
3144 \bool_if:NF \g_@@_caption_finished_bool
3145 {
3146   \bool_gset_true:N \g_@@_caption_finished_bool
3147   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3148   \int_gzero:N \c@tabularnote
3149 }
3150 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3151 \group_end:
3152 }

3153 \cs_new_protected:Npn \@_tabularnote_error:n #1
3154 {
3155   \@_error_or_warning:n { tabularnote~below~the~tabular }
3156   \@_gredirect_none:n { tabularnote~below~the~tabular }
3157 }

3158 \cs_new_protected:Npn \@_insert_tabularnotes:
3159 {
3160   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3161   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3162   \skip_vertical:N 0.65ex
```

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

```
3163 \group_begin:
3164 \l_@@_notes_code_before_tl
3165 \tl_if_empty:NF \g_@@_tabularnote_tl
3166 {
3167   \g_@@_tabularnote_tl \par
3168   \tl_gclear:N \g_@@_tabularnote_tl
3169 }
```

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.

```
3170 \int_compare:nNnT \c@tabularnote > \c_zero_int
3171 {
3172   \bool_if:NTF \l_@@_notes_para_bool
3173   {
3174     \begin { tabularnotes* }
3175     \seq_map_inline:Nn \g_@@_notes_seq
3176     { \@_one_tabularnote:nn ##1 }
3177     \strut
3178     \end { tabularnotes* }
3179   }
```

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

```
3179 \par
```

```

3180     }
3181     {
3182         \tabularnotes
3183         \seq_map_inline:Nn \g_@@_notes_seq
3184         { \@@_one_tabularnote:nn #1 }
3185         \strut
3186         \endtabularnotes
3187     }
3188 }
3189 \unskip
3190 \group_end:
3191 \bool_if:NT \l_@@_notes_bottomrule_bool
3192 {
3193     \IfPackageLoadedTF { booktabs }
3194     {

```

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

```

3195         \skip_vertical:N \aboverulesep

```

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

```

3196         { \CT@arc@ \hrule height \heavyrulewidth }
3197     }
3198     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3199 }
3200 \l_@@_notes_code_after_tl
3201 \seq_gclear:N \g_@@_notes_seq
3202 \seq_gclear:N \g_@@_notes_in_caption_seq
3203 \int_gzero:N \c@tabularnote
3204 }

```

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

```

3205 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3206 {
3207     \tl_if_novalue:nTF { #1 }
3208     { \item }
3209     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3210 }

```

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`.

```

3211 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3212 {
3213     \pgfpicture
3214     \@@_qpoint:n { row - 1 }
3215     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3216     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3217     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3218     \endpgfpicture
3219     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3220     \int_if_zero:nT \l_@@_first_row_int
3221     {
3222         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3223         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3224     }
3225     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3226 }

```

Now, the general case.

```

3227 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3228 {

```

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

```

3229 \tl_if_eq:NnT \l_@@_baseline_tl { t }
3230 { \cs_set_nopar:Npn \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`.

```

3231 \pgfpicture
3232 \@@_qpoint:n { row - 1 }
3233 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3234 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3235 {
3236   \int_set:Nn \l_tmpa_int
3237   {
3238     \str_range:Nnn
3239     \l_@@_baseline_tl
3240     6
3241     { \tl_count:o \l_@@_baseline_tl }
3242   }
3243   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3244 }
3245 {
3246   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3247   \bool_lazy_or:nnT
3248   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3249   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3250   {
3251     \@@_error:n { bad-value-for~baseline }
3252     \int_set:Nn \l_tmpa_int 1
3253   }
3254   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3255 }
3256 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3257 \endpgfpicture
3258 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3259 \int_if_zero:nT \l_@@_first_row_int
3260 {
3261   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3262   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3263 }
3264 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3265 }

```

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.

```

3266 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3267 {

```

We will compute the real width of both delimiters used.

```

3268 \dim_zero_new:N \l_@@_real_left_delim_dim
3269 \dim_zero_new:N \l_@@_real_right_delim_dim
3270 \hbox_set:Nn \l_tmpb_box
3271 {
3272   \c_math_toggle_token
3273   \left #1
3274   \vcenter
3275   {
3276     \vbox_to_ht:nn
3277     { \box_ht_plus_dp:N \l_tmpa_box }
3278     { }
3279   }
3280   \right .
3281   \c_math_toggle_token

```

```

3282     }
3283     \dim_set:Nn \l_@@_real_left_delim_dim
3284     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3285     \hbox_set:Nn \l_tmpb_box
3286     {
3287         \c_math_toggle_token
3288         \left .
3289         \vbox_to_ht:nn
3290         { \box_ht_plus_dp:N \l_tmpa_box }
3291         { }
3292         \right #2
3293         \c_math_toggle_token
3294     }
3295     \dim_set:Nn \l_@@_real_right_delim_dim
3296     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3297     \skip_horizontal:N \l_@@_left_delim_dim
3298     \skip_horizontal:N -\l_@@_real_left_delim_dim
3299     \@@_put_box_in_flow:
3300     \skip_horizontal:N \l_@@_right_delim_dim
3301     \skip_horizontal:N -\l_@@_real_right_delim_dim
3302 }

```

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).

```

3303 \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).

```

3304 {
3305     \peek_remove_spaces:n
3306     {
3307         \peek_meaning:NTF \end
3308         \@@_analyze_end:Nn
3309         {
3310             \@@_transform_preamble:

```

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

```

3311         \@@_array:o \g_@@_array_preamble_tl
3312     }
3313 }
3314 }
3315 {
3316     \@@_create_col_nodes:
3317     \endarray
3318 }

```

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

```

3319 \NewDocumentEnvironment { @@-light-syntax } { b }
3320 {

```

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`.

```

3321     \tl_if_empty:nT { #1 }

```

```

3322     { \@@_fatal:n { empty-environment } }
3323 \tl_if_in:nnT { #1 } { & }
3324     { \@@_fatal:n { ampersand-in~light-syntax } }
3325 \tl_if_in:nnT { #1 } { \ }
3326     { \@@_fatal:n { double-backslash-in~light-syntax } }

```

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`.

```

3327 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3328 }

```

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.

```

3329 {
3330   \@@_create_col_nodes:
3331   \endarray
3332 }
3333 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3334 {
3335   \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).

```

3336 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3337 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3338 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3339   \seq_set_split:Nee
3340   \seq_set_split:Non
3341   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3342 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3343 \tl_if_empty:NF \l_tmpa_tl
3344   { \seq_put_right:No \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.

```

3345 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3346   { \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` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`.

```

3347 \tl_build_begin:N \l_@@_new_body_tl
3348 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3349 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3350 \@@_line_with_light_syntax:o \l_tmpa_tl

```

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

```

3351 \seq_map_inline:Nn \l_@@_rows_seq
3352 {
3353   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3354   \@@_line_with_light_syntax:n { ##1 }
3355 }
3356 \tl_build_end:N \l_@@_new_body_tl

```

```

3357 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3358 {
3359     \int_set:Nn \l_@@_last_col_int
3360     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3361 }

```

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.

```

3362 \@@_transform_preamble:

```

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

```

3363 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3364 }
3365 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3366 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3367 {
3368     \seq_clear_new:N \l_@@_cells_seq
3369     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3370     \int_set:Nn \l_@@_nb_cols_int
3371     {
3372         \int_max:nn
3373         \l_@@_nb_cols_int
3374         { \seq_count:N \l_@@_cells_seq }
3375     }
3376     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3377     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3378     \seq_map_inline:Nn \l_@@_cells_seq
3379     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3380 }

```

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`.

```

3381 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3382 {
3383     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3384     { \@@_fatal:n { empty-environment } }

```

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

```

3385     \end { #2 }
3386 }

```

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 such as `columns-width`).

```

3387 \cs_new:Npn \@@_create_col_nodes:
3388 {
3389     \crrr
3390     \int_if_zero:nT \l_@@_first_col_int
3391     {
3392         \omit
3393         \hbox_overlap_left:n
3394         {
3395             \bool_if:NT \l_@@_code_before_bool
3396             { \pgfsys@markposition { \@@_env: - col - 0 } }
3397             \pgfpicture
3398             \pgfrememberpicturepositiononpagetrue
3399             \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3400             \str_if_empty:NF \l_@@_name_str

```

```

3401         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3402     \endpgfpicture
3403     \skip_horizontal:N 2\col@sep
3404     \skip_horizontal:N \g_@@_width_first_col_dim
3405 }
3406 &
3407 }
3408 \omit

```

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

```

3409     \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`).

```

3410     \int_if_zero:nTF \l_@@_first_col_int
3411     {
3412         \bool_if:NT \l_@@_code_before_bool
3413         {
3414             \hbox
3415             {
3416                 \skip_horizontal:N -0.5\arrayrulewidth
3417                 \pgfsys@markposition { \@@_env: - col - 1 }
3418                 \skip_horizontal:N 0.5\arrayrulewidth
3419             }
3420         }
3421         \pgfpicture
3422         \pgfrememberpicturepositiononpagetrue
3423         \pgfcoordinate { \@@_env: - col - 1 }
3424         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3425         \str_if_empty:NF \l_@@_name_str
3426         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3427         \endpgfpicture
3428     }
3429     {
3430         \bool_if:NT \l_@@_code_before_bool
3431         {
3432             \hbox
3433             {
3434                 \skip_horizontal:N 0.5\arrayrulewidth
3435                 \pgfsys@markposition { \@@_env: - col - 1 }
3436                 \skip_horizontal:N -0.5\arrayrulewidth
3437             }
3438         }
3439         \pgfpicture
3440         \pgfrememberpicturepositiononpagetrue
3441         \pgfcoordinate { \@@_env: - col - 1 }
3442         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3443         \str_if_empty:NF \l_@@_name_str
3444         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3445         \endpgfpicture
3446     }

```

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 that 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 we will add some dimensions to it.

```

3447     \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3448     \bool_if:NF \l_@@_auto_columns_width_bool
3449     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3450     {
3451         \bool_lazy_and:nnTF
3452         \l_@@_auto_columns_width_bool

```

```

3453     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3454     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3455     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3456     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3457   }
3458   \skip_horizontal:N \g_tmpa_skip
3459   \hbox
3460   {
3461     \bool_if:NT \l_@@_code_before_bool
3462     {
3463       \hbox
3464       {
3465         \skip_horizontal:N -0.5\arrayrulewidth
3466         \pgfsys@markposition { \@@_env: - col - 2 }
3467         \skip_horizontal:N 0.5\arrayrulewidth
3468       }
3469     }
3470     \pgfpicture
3471     \pgfrememberpicturepositiononpagetrue
3472     \pgfcoordinate { \@@_env: - col - 2 }
3473     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3474     \str_if_empty:NF \l_@@_name_str
3475     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3476     \endpgfpicture
3477   }

```

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.

```

3478   \int_gset_eq:NN \g_tmpa_int \c_one_int
3479   \bool_if:NTF \g_@@_last_col_found_bool
3480   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3481   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3482   {
3483     &
3484     \omit
3485     \int_gincr:N \g_tmpa_int

```

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

```

3486     \skip_horizontal:N \g_tmpa_skip
3487     \bool_if:NT \l_@@_code_before_bool
3488     {
3489       \hbox
3490       {
3491         \skip_horizontal:N -0.5\arrayrulewidth
3492         \pgfsys@markposition
3493         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3494         \skip_horizontal:N 0.5\arrayrulewidth
3495       }
3496     }

```

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

```

3497     \pgfpicture
3498     \pgfrememberpicturepositiononpagetrue
3499     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3500     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3501     \str_if_empty:NF \l_@@_name_str
3502     {
3503       \pgfnodealias
3504       { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3505       { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3506     }
3507     \endpgfpicture
3508   }

```



```

3509      &
3510      \omit

```

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

```

3511      \int_if_zero:nT \g_@@_col_total_int
3512      { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3513      \skip_horizontal:N \g_tmpa_skip
3514      \int_gincr:N \g_tmpa_int
3515      \bool_lazy_any:nF
3516      {
3517          \g_@@_delims_bool
3518          \l_@@_tabular_bool
3519          { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3520          \l_@@_exterior_arraycolsep_bool
3521          \l_@@_bar_at_end_of_pream_bool
3522      }
3523      { \skip_horizontal:N -\col@sep }
3524      \bool_if:NT \l_@@_code_before_bool
3525      {
3526          \hbox
3527          {
3528              \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.

```

3529          \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3530          { \skip_horizontal:N -\arraycolsep }
3531          \pgfsys@markposition
3532          { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3533          \skip_horizontal:N 0.5\arrayrulewidth
3534          \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3535          { \skip_horizontal:N \arraycolsep }
3536      }
3537  }
3538  \pgfpicture
3539  \pgfrememberpicturepositiononpagetrue
3540  \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3541  {
3542      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3543      {
3544          \pgfpoint
3545          { - 0.5 \arrayrulewidth - \arraycolsep }
3546          \c_zero_dim
3547      }
3548      { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3549  }
3550  \str_if_empty:NF \l_@@_name_str
3551  {
3552      \pgfnodealias
3553      { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3554      { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3555  }
3556  \endpgfpicture

3557  \bool_if:NT \g_@@_last_col_found_bool
3558  {
3559      \hbox_overlap_right:n
3560      {
3561          \skip_horizontal:N \g_@@_width_last_col_dim
3562          \skip_horizontal:N \col@sep
3563          \bool_if:NT \l_@@_code_before_bool

```

```

3564         {
3565             \pgfsys@markposition
3566             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3567         }
3568         \pgfpicture
3569         \pgfrememberpicturepositiononpagetrue
3570         \pgfcoordinate
3571         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3572         \pgfpointorigin
3573         \str_if_empty:NF \l_@@_name_str
3574         {
3575             \pgfnodealias
3576             {
3577                 \l_@@_name_str - col
3578                 - \int_eval:n { \g_@@_col_total_int + 1 }
3579             }
3580             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3581         }
3582         \endpgfpicture
3583     }
3584 }
3585 % \cr
3586 }

```

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

```

3587 \tl_const:Nn \c_@@_preamble_first_col_tl
3588 {
3589     >
3590     {

```

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).

```

3591         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3592         \bool_gset_true:N \g_@@_after_col_zero_bool
3593         \@@_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.

```

3594         \hbox_set:Nw \l_@@_cell_box
3595         \@@_math_toggle:
3596         \@@_tuning_key_small:

```

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”.

```

3597         \int_compare:nNnT \c@iRow > \c_zero_int
3598         {
3599             \bool_lazy_or:nnT
3600             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3601             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3602             {
3603                 \l_@@_code_for_first_col_tl
3604                 \xglobal \colorlet { nicematrix-first-col } { . }
3605             }
3606         }
3607     }

```

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`.

```

3608     l
3609     <
3610     {
3611         \@@_math_toggle:
3612         \hbox_set_end:

```

```

3613 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3614 \@@_adjust_size_box:
3615 \@@_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.

```

3616 \dim_gset:Nn \g_@@_width_first_col_dim
3617 { \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.

```

3618 \hbox_overlap_left:n
3619 {
3620   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3621     \@@_node_for_cell:
3622     { \box_use_drop:N \l_@@_cell_box }
3623     \skip_horizontal:N \l_@@_left_delim_dim
3624     \skip_horizontal:N \l_@@_left_margin_dim
3625     \skip_horizontal:N \l_@@_extra_left_margin_dim
3626   }
3627   \bool_gset_false:N \g_@@_empty_cell_bool
3628   \skip_horizontal:N -2\col@sep
3629 }
3630 }

```

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

```

3631 \tl_const:Nn \c_@@_preamble_last_col_tl
3632 {
3633   >
3634   {
3635     \bool_set_true:N \l_@@_in_last_col_bool

```

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

```

3636 \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.

```

3637 \bool_gset_true:N \g_@@_last_col_found_bool
3638 \int_gincr:N \c@jCol
3639 \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.

```

3640 \hbox_set:Nw \l_@@_cell_box
3641 \@@_math_toggle:
3642 \@@_tuning_key_small:

```

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”.

```

3643 \int_compare:nNnT \c@iRow > \c_zero_int
3644 {
3645   \bool_lazy_or:nnT
3646   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3647   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3648   {
3649     \l_@@_code_for_last_col_tl
3650     \xglobal \colorlet { nicematrix-last-col } { . }
3651   }
3652 }
3653 }
3654 1
3655 <
3656 {
3657   \@@_math_toggle:
3658   \hbox_set_end:
3659   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3660   \@@_adjust_size_box:
3661   \@@_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.

```

3662 \dim_gset:Nn \g_@@_width_last_col_dim
3663 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3664 \skip_horizontal:N -2\col@sep

```

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

```

3665 \hbox_overlap_right:n
3666 {
3667   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3668   {
3669     \skip_horizontal:N \l_@@_right_delim_dim
3670     \skip_horizontal:N \l_@@_right_margin_dim
3671     \skip_horizontal:N \l_@@_extra_right_margin_dim
3672     \@@_node_for_cell:
3673   }
3674 }
3675 \bool_gset_false:N \g_@@_empty_cell_bool
3676 }
3677 }

```

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

```

3678 \NewDocumentEnvironment { NiceArray } { }
3679 {
3680   \bool_gset_false:N \g_@@_delims_bool
3681   \str_if_empty:NT \g_@@_name_env_str
3682   { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

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

```

3683   \NiceArrayWithDelims . .
3684 }
3685 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3686 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3687 {
3688   \NewDocumentEnvironment { #1 NiceArray } { }
3689   {
3690     \bool_gset_true:N \g_@@_delims_bool
3691     \str_if_empty:NT \g_@@_name_env_str
3692     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3693     \@@_test_if_math_mode:
3694     \NiceArrayWithDelims #2 #3
3695   }
3696   { \endNiceArrayWithDelims }
3697 }
3698 \@@_def_env:nnn p ( )
3699 \@@_def_env:nnn b [ ]
3700 \@@_def_env:nnn B \{ \}
3701 \@@_def_env:nnn v | |
3702 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3703 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3704 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3705 {
3706   \bool_set_false:N \l_@@_preamble_bool
3707   \tl_clear:N \l_tmpa_tl
3708   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3709     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3710   \tl_put_right:Nn \l_tmpa_tl
3711     {
3712       *
3713       {
3714         \int_case:nnF \l_@@_last_col_int
3715           {
3716             { -2 } { \c@MaxMatrixCols }
3717             { -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).

```

3718       }
3719       { \int_eval:n { \l_@@_last_col_int - 1 } }
3720     }
3721     { #2 }
3722   }
3723   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3724   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3725 }
3726 \clist_map_inline:nn { p , b , B , v , V }
3727 {
3728   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3729   {
3730     \bool_gset_true:N \g_@@_delims_bool
3731     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3732     \int_if_zero:nT \l_@@_last_col_int
3733       {
3734         \bool_set_true:N \l_@@_last_col_without_value_bool
3735         \int_set:Nn \l_@@_last_col_int { -1 }
3736       }
3737     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3738     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3739   }
3740   { \use:c { end #1 NiceArray } }
3741 }

```

We define also an environment `{NiceMatrix}`

```

3742 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3743 {
3744   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3745   \int_if_zero:nT \l_@@_last_col_int
3746     {
3747       \bool_set_true:N \l_@@_last_col_without_value_bool
3748       \int_set:Nn \l_@@_last_col_int { -1 }
3749     }
3750   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3751   \bool_lazy_or:nnT
3752     { \clist_if_empty_p:N \l_@@_vlines_clist }
3753     { \l_@@_except_borders_bool }
3754     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3755   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3756 }
3757 { \endNiceArray }

```

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

```
3758 \cs_new_protected:Npn \@@_NotEmpty:
3759 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

15 {NiceTabular}, {NiceTabularX} and {NiceTabular*}

```
3760 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3761 {
```

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`.

```
3762   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3763     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3764   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3765   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3766   \tl_if_empty:NF \l_@@_short_caption_tl
3767   {
3768     \tl_if_empty:NT \l_@@_caption_tl
3769     {
3770       \@@_error_or_warning:n { short-caption-without-caption }
3771       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3772     }
3773   }
3774   \tl_if_empty:NF \l_@@_label_tl
3775   {
3776     \tl_if_empty:NT \l_@@_caption_tl
3777     { \@@_error_or_warning:n { label-without-caption } }
3778   }
3779   \NewDocumentEnvironment { TabularNote } { b }
3780   {
3781     \bool_if:NTF \l_@@_in_code_after_bool
3782     { \@@_error_or_warning:n { TabularNote~in-CodeAfter } }
3783     {
3784       \tl_if_empty:NF \g_@@_tabularnote_tl
3785       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3786       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3787     }
3788   }
3789   { }
3790   \@@_settings_for_tabular:
3791   \NiceArray { #2 }
3792 }
3793 {
3794   \endNiceArray
3795   \bool_if:NT \c_@@_testphase_table_bool
3796   { \UseTaggingSocket { tbl / hmode / end } }
3797 }
3798 \cs_new_protected:Npn \@@_settings_for_tabular:
3799 {
3800   \bool_set_true:N \l_@@_tabular_bool
3801   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3802   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3803   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3804 }
```

```
3805 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3806 {
3807   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3808   \dim_zero_new:N \l_@@_width_dim
3809   \dim_set:Nn \l_@@_width_dim { #1 }
3810   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3811   \@@_settings_for_tabular:
```

```

3812 \NiceArray { #3 }
3813 }
3814 {
3815 \endNiceArray
3816 \int_if_zero:nT \g_@@_total_X_weight_int
3817 { \@@_error:n { NiceTabularX~without~X } }
3818 }

3819 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3820 {
3821 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3822 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3823 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3824 \@@_settings_for_tabular:
3825 \NiceArray { #3 }
3826 }
3827 { \endNiceArray }

```

16 After the construction of the array

The following command will be used when the key `rounded-corners` is in force (this is the key `rounded-corners` for the whole environment and *not* the key `rounded-corners` of a command `\Block`).

```

3828 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3829 {
3830 \bool_lazy_all:nT
3831 {
3832 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3833 \l_@@_hvlines_bool
3834 { ! \g_@@_delims_bool }
3835 { ! \l_@@_except_borders_bool }
3836 }
3837 {
3838 \bool_set_true:N \l_@@_except_borders_bool
3839 \clist_if_empty:NF \l_@@_corners_clist
3840 { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3841 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3842 {
3843 \@@_stroke_block:nnn
3844 {
3845 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3846 draw = \l_@@_rules_color_tl
3847 }
3848 { 1-1 }
3849 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3850 }
3851 }
3852 }

3853 \cs_new_protected:Npn \@@_after_array:
3854 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_ii:` in order to come back to the standard definition of `\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```

3855 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3856 \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.

```

3857     \bool_if:NT \g_@@_last_col_found_bool
3858     { \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`.

```

3859     \bool_if:NT \l_@@_last_col_without_value_bool
3860     { \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.

```

3861     \bool_if:NT \l_@@_last_row_without_value_bool
3862     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3863     \tl_gput_right:Ne \g_@@_aux_tl
3864     {
3865       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3866       {
3867         \int_use:N \l_@@_first_row_int ,
3868         \int_use:N \c@iRow ,
3869         \int_use:N \g_@@_row_total_int ,
3870         \int_use:N \l_@@_first_col_int ,
3871         \int_use:N \c@jCol ,
3872         \int_use:N \g_@@_col_total_int
3873       }
3874     }

```

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`).

```

3875     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3876     {
3877       \tl_gput_right:Ne \g_@@_aux_tl
3878       {
3879         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3880         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3881       }
3882     }
3883     \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3884     {
3885       \tl_gput_right:Ne \g_@@_aux_tl
3886       {
3887         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3888         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3889         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3890         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3891       }
3892     }

```

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

```

3893     \@@_create_diag_nodes:

```

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

```

3894     \pgfpicture
3895     \int_step_inline:nn \c@iRow
3896     {
3897       \pgfnodealias
3898       { \@@_env: - ##1 - last }
3899       { \@@_env: - ##1 - \int_use:N \c@jCol }

```



```

3900     }
3901     \int_step_inline:nn \c@jCol
3902     {
3903         \pgfnodealias
3904         { \l_@@_env: - last - ##1 }
3905         { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3906     }
3907     \str_if_empty:NF \l_@@_name_str
3908     {
3909         \int_step_inline:nn \c@iRow
3910         {
3911             \pgfnodealias
3912             { \l_@@_name_str - ##1 - last }
3913             { \l_@@_env: - ##1 - \int_use:N \c@jCol }
3914         }
3915         \int_step_inline:nn \c@jCol
3916         {
3917             \pgfnodealias
3918             { \l_@@_name_str - last - ##1 }
3919             { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3920         }
3921     }
3922     \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.

```

3923     \bool_if:NT \l_@@_parallelize_diags_bool
3924     {
3925         \int_gzero_new:N \g_@@_ddots_int
3926         \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.

```

3927         \dim_gzero_new:N \g_@@_delta_x_one_dim
3928         \dim_gzero_new:N \g_@@_delta_y_one_dim
3929         \dim_gzero_new:N \g_@@_delta_x_two_dim
3930         \dim_gzero_new:N \g_@@_delta_y_two_dim
3931     }
3932     \int_zero_new:N \l_@@_initial_i_int
3933     \int_zero_new:N \l_@@_initial_j_int
3934     \int_zero_new:N \l_@@_final_i_int
3935     \int_zero_new:N \l_@@_final_j_int
3936     \bool_set_false:N \l_@@_initial_open_bool
3937     \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.

```

3938     \bool_if:NT \l_@@_small_bool
3939     {
3940         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3941         \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.

```

3942         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3943         { 0.6 \l_@@_xdots_shorten_start_dim }
3944         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

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

```

3945         { 0.6 \l_@@_xdots_shorten_end_dim }
3946     }

```

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

```

3947 \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_cells_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3948 \@@_compute_corners:

```

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

```

3949 \@@_adjust_pos_of_blocks_seq:
3950 \@@_deal_with_rounded_corners:
3951 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3952 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3953 \IfPackageLoadedT { tikz }
3954 {
3955     \tikzset
3956     {
3957         every-picture / .style =
3958         {
3959             overlay ,
3960             remember~picture ,
3961             name~prefix = \@@_env: -
3962         }
3963     }
3964 }
3965 \bool_if:NT \c_@@_tagging_array_bool
3966 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3967 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3968 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3969 \cs_set_eq:NN \OverBrace \@@_OverBrace
3970 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3971 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3972 \cs_set_eq:NN \line \@@_line
3973 \g_@@_pre_code_after_tl
3974 \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.

```

3975 \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).

```

3976 \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`).

```

3977 \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
3978 { \@@_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:`.

```

3979 \bool_set_true:N \l_@@_in_code_after_bool
3980 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3981 \scan_stop:
3982 \tl_gclear:N \g_nicematrix_code_after_tl
3983 \group_end:

```

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

```

3984 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3985 \tl_if_empty:NF \g_@@_pre_code_before_tl
3986 {
3987   \tl_gput_right:Ne \g_@@_aux_tl
3988   {
3989     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3990     { \exp_not:o \g_@@_pre_code_before_tl }
3991   }
3992   \tl_gclear:N \g_@@_pre_code_before_tl
3993 }
3994 \tl_if_empty:NF \g_nicematrix_code_before_tl
3995 {
3996   \tl_gput_right:Ne \g_@@_aux_tl
3997   {
3998     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3999     { \exp_not:o \g_nicematrix_code_before_tl }
4000   }
4001   \tl_gclear:N \g_nicematrix_code_before_tl
4002 }

4003 \str_gclear:N \g_@@_name_env_str
4004 \@@_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`.

```

4005 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4006 }

```

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`.

```

4007 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4008 { \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).

```

4009 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4010 {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

4011 \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4012 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4013 }

```

The following command must *not* be protected.

```

4014 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4015 {
4016   { #1 }
4017   { #2 }
4018   {
4019     \int_compare:nNnTF { #3 } > { 99 }
4020     { \int_use:N \c@iRow }
4021     { #3 }
4022   }
4023   {
4024     \int_compare:nNnTF { #4 } > { 99 }
4025     { \int_use:N \c@jCol }
4026     { #4 }
4027   }
4028   { #5 }
4029 }

```

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).

```

4030 \hook_gput_code:nnn { begindocument } { . }
4031 {
4032   \cs_new_protected:Npe \@@_draw_dotted_lines:
4033   {
4034     \c_@@_pgfortikzpicture_tl
4035     \@@_draw_dotted_lines_i:
4036     \c_@@_endpgfortikzpicture_tl
4037   }
4038 }

```

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

```

4039 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4040 {
4041   \pgfrememberpicturepositiononpagetrue
4042   \pgfrelevantforpicturesizefalse
4043   \g_@@_HVdotsfor_lines_tl
4044   \g_@@_Vdots_lines_tl
4045   \g_@@_Ddots_lines_tl
4046   \g_@@_Iddots_lines_tl
4047   \g_@@_Cdots_lines_tl
4048   \g_@@_Ldots_lines_tl
4049 }

4050 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4051 {
4052   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4053   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4054 }

```

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

```

4055 \pgfdeclareshape { @@_diag_node }
4056 {
4057   \savedanchor { \five }
4058   {
4059     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

4060     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4061 }
4062 \anchor { 5 } { \five }
4063 \anchor { center } { \pgfpointorigin }
4064 \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4065 \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4066 \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4067 \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4068 \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4069 \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4070 \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4071 \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4072 }

```

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).

```

4073 \cs_new_protected:Npn \@@_create_diag_nodes:
4074 {
4075   \pgfpicture
4076   \pgfrememberpicturerepositiononpagetrue
4077   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4078   {
4079     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4080     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4081     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4082     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4083     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4084     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4085     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4086     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4087     \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@@_diag_node`) that we will construct.

```

4088     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4089     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4090     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4091     \str_if_empty:NF \l_@@_name_str
4092     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4093   }

```

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

```

4094     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4095     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4096     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4097     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4098     \pgfcoordinate
4099     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4100     \pgfnodealias
4101     { \@@_env: - last }
4102     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4103     \str_if_empty:NF \l_@@_name_str
4104     {
4105       \pgfnodealias
4106       { \l_@@_name_str - \int_use:N \l_tmpa_int }
4107       { \@@_env: - \int_use:N \l_tmpa_int }
4108       \pgfnodealias
4109       { \l_@@_name_str - last }
4110       { \@@_env: - last }
4111     }
4112   \endpgfpicture
4113 }

```

17 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.

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

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.

```
4114 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4115 {
```

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

```
4116 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4117 \int_set:Nn \l_@@_initial_i_int { #1 }
4118 \int_set:Nn \l_@@_initial_j_int { #2 }
4119 \int_set:Nn \l_@@_final_i_int { #1 }
4120 \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.

```
4121 \bool_set_false:N \l_@@_stop_loop_bool
4122 \bool_do_until:Nn \l_@@_stop_loop_bool
4123 {
4124   \int_add:Nn \l_@@_final_i_int { #3 }
4125   \int_add:Nn \l_@@_final_j_int { #4 }
4126   \bool_set_false:N \l_@@_final_open_bool
```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```
4127 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4128   \if_int_compare:w #3 = \c_one_int
4129     \bool_set_true:N \l_@@_final_open_bool
4130   \else:
4131     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4132       \bool_set_true:N \l_@@_final_open_bool
4133     \fi:
4134   \fi:
4135 \else:
4136   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
```

```

4137         \if_int_compare:w #4 = -1
4138         \bool_set_true:N \l_@@_final_open_bool
4139         \fi:
4140     \else:
4141         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4142         \if_int_compare:w #4 = \c_one_int
4143         \bool_set_true:N \l_@@_final_open_bool
4144         \fi:
4145     \fi:
4146 \fi:
4147 \fi:
4148 \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.

```

4149     {

```

We do a step backwards.

```

4150         \int_sub:Nn \l_@@_final_i_int { #3 }
4151         \int_sub:Nn \l_@@_final_j_int { #4 }
4152         \bool_set_true:N \l_@@_stop_loop_bool
4153     }

```

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`.

```

4154     {
4155         \cs_if_exist:cTF
4156         {
4157             @@ _ dotted _
4158             \int_use:N \l_@@_final_i_int -
4159             \int_use:N \l_@@_final_j_int
4160         }
4161         {
4162             \int_sub:Nn \l_@@_final_i_int { #3 }
4163             \int_sub:Nn \l_@@_final_j_int { #4 }
4164             \bool_set_true:N \l_@@_final_open_bool
4165             \bool_set_true:N \l_@@_stop_loop_bool
4166         }
4167         {
4168             \cs_if_exist:cTF
4169             {
4170                 pgf @ sh @ ns @ \@@_env:
4171                 - \int_use:N \l_@@_final_i_int
4172                 - \int_use:N \l_@@_final_j_int
4173             }
4174             { \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.

```

4175         {
4176             \cs_set:cpn
4177             {
4178                 @@ _ dotted _
4179                 \int_use:N \l_@@_final_i_int -
4180                 \int_use:N \l_@@_final_j_int
4181             }
4182             { }
4183         }
4184     }
4185 }
4186 }

```

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

```
4187 \bool_set_false:N \l_@@_stop_loop_bool
```

The following line of code is only for efficiency in the following loop.

```
4188 \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4189 \bool_do_until:Nn \l_@@_stop_loop_bool
4190 {
4191   \int_sub:Nn \l_@@_initial_i_int { #3 }
4192   \int_sub:Nn \l_@@_initial_j_int { #4 }
4193   \bool_set_false:N \l_@@_initial_open_bool
```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```
4194 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4195 \if_int_compare:w #3 = \c_one_int
4196 \bool_set_true:N \l_@@_initial_open_bool
4197 \else:
```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```
4198 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4199 \bool_set_true:N \l_@@_initial_open_bool
4200 \fi:
4201 \fi:
4202 \else:
4203 \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4204 \if_int_compare:w #4 = \c_one_int
4205 \bool_set_true:N \l_@@_initial_open_bool
4206 \fi:
4207 \else:
4208 \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4209 \if_int_compare:w #4 = -1
4210 \bool_set_true:N \l_@@_initial_open_bool
4211 \fi:
4212 \fi:
4213 \fi:
4214 \fi:
4215 \bool_if:NTF \l_@@_initial_open_bool
4216 {
4217   \int_add:Nn \l_@@_initial_i_int { #3 }
4218   \int_add:Nn \l_@@_initial_j_int { #4 }
4219   \bool_set_true:N \l_@@_stop_loop_bool
4220 }
4221 {
4222   \cs_if_exist:cTF
4223   {
4224     @@ _ dotted _
4225     \int_use:N \l_@@_initial_i_int -
4226     \int_use:N \l_@@_initial_j_int
4227   }
4228   {
4229     \int_add:Nn \l_@@_initial_i_int { #3 }
4230     \int_add:Nn \l_@@_initial_j_int { #4 }
4231     \bool_set_true:N \l_@@_initial_open_bool
4232     \bool_set_true:N \l_@@_stop_loop_bool
4233   }
4234   {
4235     \cs_if_exist:cTF
4236     {
4237       pgf @ sh @ ns @ \@@_env:
4238       - \int_use:N \l_@@_initial_i_int
4239       - \int_use:N \l_@@_initial_j_int
4240     }
```



```

4241         { \bool_set_true:N \l_@@_stop_loop_bool }
4242         {
4243             \cs_set:cpn
4244             {
4245                 @@ _ dotted _
4246                 \int_use:N \l_@@_initial_i_int -
4247                 \int_use:N \l_@@_initial_j_int
4248             }
4249             { }
4250         }
4251     }
4252 }
4253 }

```

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.

```

4254     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4255     {
4256         { \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`.

```

4257         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4258         { \int_use:N \l_@@_final_i_int }
4259         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4260         { } % for the name of the block
4261     }
4262 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analyse of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4263 \cs_new_protected:Npn \@@_open_shorten:
4264 {
4265     \bool_if:NT \l_@@_initial_open_bool
4266     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4267     \bool_if:NT \l_@@_final_open_bool
4268     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4269 }

```

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).

```

4270 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4271 {
4272     \int_set_eq:NN \l_@@_row_min_int \c_one_int
4273     \int_set_eq:NN \l_@@_col_min_int \c_one_int
4274     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4275     \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`.

```

4276     \seq_if_empty:NF \g_@@_submatrix_seq
4277     {
4278         \seq_map_inline:Nn \g_@@_submatrix_seq
4279         { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4280     }
4281 }

```

#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.

Here is the programming of that command with the the standard syntax of L3.

```
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}
```

However, for efficiency, we will use the following version.

```
4282 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4283 {
4284   \if_int_compare:w #3 > #1
4285   \else:
4286     \if_int_compare:w #1 > #5
4287     \else:
4288       \if_int_compare:w #4 > #2
4289       \else:
4290         \if_int_compare:w #2 > #6
4291         \else:
4292           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4293           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4294           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4295           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4296         \fi:
4297       \fi:
4298     \fi:
4299   \fi:
4300 }

4301 \cs_new_protected:Npn \@@_set_initial_coords:
4302 {
4303   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4304   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4305 }
4306 \cs_new_protected:Npn \@@_set_final_coords:
4307 {
4308   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4309   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4310 }
4311 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4312 {
4313   \pgfpointanchor
4314   {
4315     \@@_env:
4316     - \int_use:N \l_@@_initial_i_int
4317     - \int_use:N \l_@@_initial_j_int
4318   }
4319   { #1 }
4320   \@@_set_initial_coords:
```

```

4321 }
4322 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4323 {
4324   \pgfpointanchor
4325   {
4326     \@@_env:
4327     - \int_use:N \l_@@_final_i_int
4328     - \int_use:N \l_@@_final_j_int
4329   }
4330   { #1 }
4331   \@@_set_final_coords:
4332 }
4333 \cs_new_protected:Npn \@@_open_x_initial_dim:
4334 {
4335   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4336   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4337   {
4338     \cs_if_exist:cT
4339     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4340     {
4341       \pgfpointanchor
4342       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4343       { west }
4344       \dim_set:Nn \l_@@_x_initial_dim
4345       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4346     }
4347   }

```

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

```

4348   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4349   {
4350     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4351     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4352     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4353   }
4354 }
4355 \cs_new_protected:Npn \@@_open_x_final_dim:
4356 {
4357   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4358   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4359   {
4360     \cs_if_exist:cT
4361     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4362     {
4363       \pgfpointanchor
4364       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4365       { east }
4366       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4367       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4368     }
4369   }

```

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

```

4370   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4371   {
4372     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4373     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4374     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4375   }
4376 }

```

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.

```

4377 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4378 {
4379   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4380   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4381   {
4382     \@@_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.

```

4383   \group_begin:
4384   \@@_open_shorten:
4385   \int_if_zero:nTF { #1 }
4386   { \color { nicematrix-first-row } }
4387   {

```

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.

```

4388     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4389     { \color { nicematrix-last-row } }
4390   }
4391   \keys_set:nn { nicematrix / xdots } { #3 }
4392   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4393   \@@_actually_draw_Ldots:
4394   \group_end:
4395 }
4396 }

```

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`.

```

4397 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4398 {
4399   \bool_if:NTF \l_@@_initial_open_bool
4400   {
4401     \@@_open_x_initial_dim:
4402     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4403     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4404   }
4405   { \@@_set_initial_coords_from_anchor:n { base-east } }
4406   \bool_if:NTF \l_@@_final_open_bool
4407   {
4408     \@@_open_x_final_dim:
4409     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4410     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4411   }
4412   { \@@_set_final_coords_from_anchor:n { base-west } }

```

Now the case of a `\Hdotsfor` (or when there is only a `\Ldots`) in the “last row” (that case will probably arise when the final user draws an arrow to indicate the number of columns of the matrix). In the “first row”, we don’t need any adjustment.

```

4413   \bool_lazy_all:nTF
4414   {
4415     \l_@@_initial_open_bool
4416     \l_@@_final_open_bool

```

```

4417     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4418   }
4419   {
4420     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4421     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4422   }

```

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 (?).

```

4423   {
4424     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4425     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4426   }
4427   \@@_draw_line:
4428 }

```

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.

```

4429 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4430 {
4431   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4432   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4433   {
4434     \@@_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.

```

4435   \group_begin:
4436     \@@_open_shorten:
4437     \int_if_zero:nTF { #1 }
4438       { \color { nicematrix-first-row } }
4439     {

```

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.

```

4440       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4441       { \color { nicematrix-last-row } }
4442     }
4443     \keys_set:nn { nicematrix / xdots } { #3 }
4444     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4445     \@@_actually_draw_Cdots:
4446   \group_end:
4447 }
4448 }

```

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`.

```

4449 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4450 {
4451   \bool_if:NTF \l_@@_initial_open_bool
4452     { \@@_open_x_initial_dim: }
4453     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4454   \bool_if:NTF \l_@@_final_open_bool

```

```

4455     { \@@_open_x_final_dim: }
4456     { \@@_set_final_coords_from_anchor:n { mid~west } }
4457 \bool_lazy_and:nnTF
4458     \l_@@_initial_open_bool
4459     \l_@@_final_open_bool
4460     {
4461         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4462         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4463         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4464         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4465         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4466     }
4467     {
4468         \bool_if:NT \l_@@_initial_open_bool
4469         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4470         \bool_if:NT \l_@@_final_open_bool
4471         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4472     }
4473     \@@_draw_line:
4474 }

4475 \cs_new_protected:Npn \@@_open_y_initial_dim:
4476 {
4477     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4478     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4479     {
4480         \cs_if_exist:cT
4481         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4482         {
4483             \pgfpointanchor
4484             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4485             { north }
4486             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4487             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4488         }
4489     }
4490     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4491     {
4492         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4493         \dim_set:Nn \l_@@_y_initial_dim
4494         {
4495             \fp_to_dim:n
4496             {
4497                 \pgf@y
4498                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4499             }
4500         }
4501     }
4502 }

4503 \cs_new_protected:Npn \@@_open_y_final_dim:
4504 {
4505     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4506     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4507     {
4508         \cs_if_exist:cT
4509         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4510         {
4511             \pgfpointanchor
4512             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4513             { south }
4514             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4515             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4516         }
4517     }

```

```

4518 \dim_compare:nNtT \l_@@_y_final_dim = \c_max_dim
4519 {
4520   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4521   \dim_set:Nn \l_@@_y_final_dim
4522     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4523 }
4524 }

```

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.

```

4525 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4526 {
4527   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4528   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4529   {
4530     \@@_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.

```

4531 \group_begin:
4532 \@@_open_shorten:
4533 \int_if_zero:nTF { #2 }
4534 { \color { nicematrix-first-col } }
4535 {
4536   \int_compare:nNtT { #2 } = \l_@@_last_col_int
4537   { \color { nicematrix-last-col } }
4538 }
4539 \keys_set:nn { nicematrix / xdots } { #3 }
4540 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4541 \@@_actually_draw_Vdots:
4542 \group_end:
4543 }
4544 }

```

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`.

```

4545 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4546 {

```

First, the case of a dotted line open on both sides.

```

4547 \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool

```

We have to determine the x -value of the vertical rule that we will have to draw.

```

4548 {
4549   \@@_open_y_initial_dim:
4550   \@@_open_y_final_dim:
4551   \int_if_zero:nTF \l_@@_initial_j_int

```

We have a dotted line open on both sides in the “first column”.

```

4552 {
4553   \@@_qpoint:n { col - 1 }
4554   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4555   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim

```

```

4556     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4557     \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4558   }
4559   {
4560     \bool_lazy_and:nnTF
4561     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4562     { \int_compare_p:nNn \l_@@_initial_j_int = \g_@@_col_total_int }

```

We have a dotted line open on both sides in the “last column”.

```

4563   {
4564     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4565     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4566     \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4567     \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4568     \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4569   }

```

We have a dotted line open on both sides which is *not* in an exterior column.

```

4570   {
4571     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4572     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4573     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4574     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4575   }
4576 }
4577 }

```

Now, the dotted line is *not* open on both sides (maybe open on only one side).

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

```

4578   {
4579     \bool_set_false:N \l_tmpa_bool
4580     \bool_if:NF \l_@@_initial_open_bool
4581     {
4582       \bool_if:NF \l_@@_final_open_bool
4583       {
4584         \@@_set_initial_coords_from_anchor:n { south-west }
4585         \@@_set_final_coords_from_anchor:n { north-west }
4586         \bool_set:Nn \l_tmpa_bool
4587         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4588       }
4589     }

```

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

```

4590     \bool_if:NTF \l_@@_initial_open_bool
4591     {
4592       \@@_open_y_initial_dim:
4593       \@@_set_final_coords_from_anchor:n { north }
4594       \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4595     }
4596     {
4597       \@@_set_initial_coords_from_anchor:n { south }
4598       \bool_if:NTF \l_@@_final_open_bool
4599       \@@_open_y_final_dim:

```

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.

```

4600     {
4601       \@@_set_final_coords_from_anchor:n { north }
4602       \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4603       {
4604         \dim_set:Nn \l_@@_x_initial_dim
4605         {
4606           \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4607           \l_@@_x_initial_dim \l_@@_x_final_dim
4608         }

```



```

4609         }
4610     }
4611 }
4612 }
4613 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4614 \@@_draw_line:
4615 }

```

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.

```

4616 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4617 {
4618     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4619     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4620     {
4621         \@@_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.

```

4622     \group_begin:
4623     \@@_open_shorten:
4624     \keys_set:nn { nicematrix / xdots } { #3 }
4625     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4626     \@@_actually_draw_Ddots:
4627     \group_end:
4628 }
4629 }

```

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`.

```

4630 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4631 {
4632     \bool_if:NTF \l_@@_initial_open_bool
4633     {
4634         \@@_open_y_initial_dim:
4635         \@@_open_x_initial_dim:
4636     }
4637     { \@@_set_initial_coords_from_anchor:n { south-east } }
4638     \bool_if:NTF \l_@@_final_open_bool
4639     {
4640         \@@_open_x_final_dim:
4641         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4642     }
4643     { \@@_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.

```

4644     \bool_if:NT \l_@@_parallelize_diags_bool
4645     {
4646         \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).

```
4647 \int_compare:nNnTF \g_@@_ddots_int = \c_one_int
```

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.

```
4648 {
4649   \dim_gset:Nn \g_@@_delta_x_one_dim
4650   { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4651   \dim_gset:Nn \g_@@_delta_y_one_dim
4652   { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4653 }
```

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`.

```
4654 {
4655   \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4656   {
4657     \dim_set:Nn \l_@@_y_final_dim
4658     {
4659       \l_@@_y_initial_dim +
4660       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4661       \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4662     }
4663   }
4664 }
4665 }
4666 \@@_draw_line:
4667 }
```

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.

```
4668 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4669 {
4670   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4671   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4672   {
4673     \@@_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.

```
4674   \group_begin:
4675   \@@_open_shorten:
4676   \keys_set:nn { nicematrix / xdots } { #3 }
4677   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4678   \@@_actually_draw_Iddots:
4679   \group_end:
4680 }
4681 }
```

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`.

```

4682 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4683 {
4684   \bool_if:NTF \l_@@_initial_open_bool
4685   {
4686     \@@_open_y_initial_dim:
4687     \@@_open_x_initial_dim:
4688   }
4689   { \@@_set_initial_coords_from_anchor:n { south-west } }
4690   \bool_if:NTF \l_@@_final_open_bool
4691   {
4692     \@@_open_y_final_dim:
4693     \@@_open_x_final_dim:
4694   }
4695   { \@@_set_final_coords_from_anchor:n { north-east } }
4696   \bool_if:NT \l_@@_parallelize_diags_bool
4697   {
4698     \int_gincr:N \g_@@_iddots_int
4699     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4700     {
4701       \dim_gset:Nn \g_@@_delta_x_two_dim
4702       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4703       \dim_gset:Nn \g_@@_delta_y_two_dim
4704       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4705     }
4706     {
4707       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4708       {
4709         \dim_set:Nn \l_@@_y_final_dim
4710         {
4711           \l_@@_y_initial_dim +
4712           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4713           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4714         }
4715       }
4716     }
4717   }
4718   \@@_draw_line:
4719 }

```

18 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`

```

4720 \cs_new_protected:Npn \@@_draw_line:
4721 {
4722   \pgfrememberpicturepositiononpagetrue
4723   \pgf@relevantforpicturesizefalse

```

```

4724 \bool_lazy_or:nnTF
4725 { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4726 \l_@@_dotted_bool
4727 \@@_draw_standard_dotted_line:
4728 \@@_draw_unstandard_dotted_line:
4729 }

```

We have to do a special construction with `\exp_args:No` to be able to put in the list of options in the correct place in the Tikz instruction.

```

4730 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4731 {
4732   \begin { scope }
4733   \@@_draw_unstandard_dotted_line:o
4734   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4735 }

```

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.

```

4736 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4737 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4738 {
4739   \@@_draw_unstandard_dotted_line:nooo
4740   { #1 }
4741   \l_@@_xdots_up_tl
4742   \l_@@_xdots_down_tl
4743   \l_@@_xdots_middle_tl
4744 }

```

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

```

4745 \hook_gput_code:nnn { begindocument } { . }
4746 {
4747   \IfPackageLoadedT { tikz }
4748   {
4749     \tikzset
4750     {
4751       @@_node_above / .style = { sloped , above } ,
4752       @@_node_below / .style = { sloped , below } ,
4753       @@_node_middle / .style =
4754       {
4755         sloped ,
4756         inner~sep = \c_@@_innersep_middle_dim
4757       }
4758     }
4759   }
4760 }

4761 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4762 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4763 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shorten >` and `shorten <` of TikZ in the command `\draw`, we don't have the expected output with `{decorate,decoration=brace}` is used.

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.

```

4764 \dim_zero_new:N \l_@@_l_dim
4765 \dim_set:Nn \l_@@_l_dim
4766 {
4767   \fp_to_dim:n

```

```

4768     {
4769         sqrt
4770         (
4771             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4772             +
4773             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4774         )
4775     }
4776 }

```

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.

```

4777 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4778 {
4779     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4780     \@@_draw_unstandard_dotted_line_i:
4781 }

```

If the key `xdots/horizontal-labels` has been used.

```

4782 \bool_if:NT \l_@@_xdots_h_labels_bool
4783 {
4784     \tikzset
4785     {
4786         @@_node_above / .style = { auto = left } ,
4787         @@_node_below / .style = { auto = right } ,
4788         @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4789     }
4790 }
4791 \tl_if_empty:nF { #4 }
4792 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4793 \draw
4794 [ #1 ]
4795 ( \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).

```

4796 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4797 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4798 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4799 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4800 \end { scope }
4801 }
4802 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4803 {
4804     \dim_set:Nn \l_tmpa_dim
4805     {
4806         \l_@@_x_initial_dim
4807         + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4808         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4809     }
4810     \dim_set:Nn \l_tmpb_dim
4811     {
4812         \l_@@_y_initial_dim
4813         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4814         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4815     }
4816     \dim_set:Nn \l_@@_tmpc_dim
4817     {
4818         \l_@@_x_final_dim
4819         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4820         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4821     }

```

```

4822 \dim_set:Nn \l_@@_tmpd_dim
4823 {
4824   \l_@@_y_final_dim
4825   - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4826   * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4827 }
4828 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4829 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4830 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4831 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4832 }

```

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

```

4833 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4834 {
4835   \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.

```

4836   \dim_zero_new:N \l_@@_l_dim
4837   \dim_set:Nn \l_@@_l_dim
4838   {
4839     \fp_to_dim:n
4840     {
4841       sqrt
4842       (
4843         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4844         +
4845         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4846       )
4847     }
4848   }

```

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.

```

4849   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4850   {
4851     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4852     \@@_draw_standard_dotted_line_i:
4853   }
4854   \group_end:
4855   \bool_lazy_all:nF
4856   {
4857     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4858     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4859     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4860   }
4861   \l_@@_labels_standard_dotted_line:
4862 }
4863 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4864 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4865 {

```

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

```

4866   \int_set:Nn \l_tmpa_int
4867   {
4868     \dim_ratio:nn
4869     {
4870       \l_@@_l_dim

```

```

4871         - \l_@@_xdots_shorten_start_dim
4872         - \l_@@_xdots_shorten_end_dim
4873     }
4874     \l_@@_xdots_inter_dim
4875 }

```

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

```

4876 \dim_set:Nn \l_tmpa_dim
4877 {
4878   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4879   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4880 }
4881 \dim_set:Nn \l_tmpb_dim
4882 {
4883   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4884   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4885 }

```

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.

```

4886 \dim_gadd:Nn \l_@@_x_initial_dim
4887 {
4888   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4889   \dim_ratio:nn
4890   {
4891     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4892     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4893   }
4894   { 2 \l_@@_l_dim }
4895 }
4896 \dim_gadd:Nn \l_@@_y_initial_dim
4897 {
4898   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4899   \dim_ratio:nn
4900   {
4901     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4902     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4903   }
4904   { 2 \l_@@_l_dim }
4905 }
4906 \pgf@relevantforpicturesizefalse
4907 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4908 {
4909   \pgfpathcircle
4910   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4911   { \l_@@_xdots_radius_dim }
4912   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4913   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4914 }
4915 \pgfusepathqfill
4916 }

```

```

4917 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4918 {
4919   \pgfscope
4920   \pgftransformshift
4921   {
4922     \pgfpointlineattime { 0.5 }
4923     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4924     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4925   }
4926   \fp_set:Nn \l_tmpa_fp

```

```

4927 {
4928   atand
4929   (
4930     \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4931     \l_@@_x_final_dim - \l_@@_x_initial_dim
4932   )
4933 }
4934 \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4935 \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4936 \tl_if_empty:NF \l_@@_xdots_middle_tl
4937 {
4938   \begin { pgfscope }
4939   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4940   \pgfnode
4941   { rectangle }
4942   { center }
4943   {
4944     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4945     {
4946       \c_math_toggle_token
4947       \scriptstyle \l_@@_xdots_middle_tl
4948       \c_math_toggle_token
4949     }
4950   }
4951   { }
4952   {
4953     \pgfsetfillcolor { white }
4954     \pgfusepath { fill }
4955   }
4956   \end { pgfscope }
4957 }
4958 \tl_if_empty:NF \l_@@_xdots_up_tl
4959 {
4960   \pgfnode
4961   { rectangle }
4962   { south }
4963   {
4964     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4965     {
4966       \c_math_toggle_token
4967       \scriptstyle \l_@@_xdots_up_tl
4968       \c_math_toggle_token
4969     }
4970   }
4971   { }
4972   { \pgfusepath { } }
4973 }
4974 \tl_if_empty:NF \l_@@_xdots_down_tl
4975 {
4976   \pgfnode
4977   { rectangle }
4978   { north }
4979   {
4980     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4981     {
4982       \c_math_toggle_token
4983       \scriptstyle \l_@@_xdots_down_tl
4984       \c_math_toggle_token
4985     }
4986   }
4987   { }
4988   { \pgfusepath { } }
4989 }

```



```

4990 \endpgfscope
4991 }

```

19 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.

```

4992 \hook_gput_code:nnn { begindocument } { . }
4993 {
4994   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4995   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4996   \cs_new_protected:Npn \@@_Ldots
4997     { \@@_collect_options:n { \@@_Ldots_i } }
4998   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4999     {
5000       \int_if_zero:nTF \c@jCol
5001       { \@@_error:nn { in~first~col } \Ldots }
5002       {
5003         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5004         { \@@_error:nn { in~last~col } \Ldots }
5005         {
5006           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
5007           { #1 , down = #2 , up = #3 , middle = #4 }
5008         }
5009       }
5010       \bool_if:NF \l_@@_nullify_dots_bool
5011       { \phantom { \ensuremath { \@@_old_ldots } } }
5012       \bool_gset_true:N \g_@@_empty_cell_bool
5013     }

5014   \cs_new_protected:Npn \@@_Cdots
5015     { \@@_collect_options:n { \@@_Cdots_i } }
5016   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5017     {
5018       \int_if_zero:nTF \c@jCol
5019       { \@@_error:nn { in~first~col } \Cdots }
5020       {
5021         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5022         { \@@_error:nn { in~last~col } \Cdots }
5023         {
5024           \@@_instruction_of_type:nnn \c_false_bool { Cdots }
5025           { #1 , down = #2 , up = #3 , middle = #4 }
5026         }
5027       }
5028       \bool_if:NF \l_@@_nullify_dots_bool
5029       { \phantom { \ensuremath { \@@_old_cdots } } }
5030       \bool_gset_true:N \g_@@_empty_cell_bool
5031     }

```

```

5032 \cs_new_protected:Npn \@@_Vdots
5033 { \@@_collect_options:n { \@@_Vdots_i } }
5034 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5035 {
5036   \int_if_zero:nTF \c@iRow
5037   { \@@_error:nn { in~first~row } \Vdots }
5038   {
5039     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5040     { \@@_error:nn { in~last~row } \Vdots }
5041     {
5042       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5043       { #1 , down = #2 , up = #3 , middle = #4 }
5044     }
5045   }
5046   \bool_if:NF \l_@@_nullify_dots_bool
5047   { \phantom { \ensuremath { \@@_old_vdots } } }
5048   \bool_gset_true:N \g_@@_empty_cell_bool
5049 }

5050 \cs_new_protected:Npn \@@_Ddots
5051 { \@@_collect_options:n { \@@_Ddots_i } }
5052 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5053 {
5054   \int_case:nnF \c@iRow
5055   {
5056     0 { \@@_error:nn { in~first~row } \Ddots }
5057     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5058   }
5059   {
5060     \int_case:nnF \c@jCol
5061     {
5062       0 { \@@_error:nn { in~first~col } \Ddots }
5063       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5064     }
5065     {
5066       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5067       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5068       { #1 , down = #2 , up = #3 , middle = #4 }
5069     }
5070   }
5071 }
5072 \bool_if:NF \l_@@_nullify_dots_bool
5073 { \phantom { \ensuremath { \@@_old_ddots } } }
5074 \bool_gset_true:N \g_@@_empty_cell_bool
5075 }

5076 \cs_new_protected:Npn \@@_Iddots
5077 { \@@_collect_options:n { \@@_Iddots_i } }
5078 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5079 {
5080   \int_case:nnF \c@iRow
5081   {
5082     0 { \@@_error:nn { in~first~row } \Iddots }
5083     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5084   }
5085   {
5086     \int_case:nnF \c@jCol
5087     {
5088       0 { \@@_error:nn { in~first~col } \Iddots }
5089       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5090     }
5091     {

```

```

5092         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5093         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5094         { #1 , down = #2 , up = #3 , middle = #4 }
5095     }
5096 }
5097 \bool_if:NF \l_@@_nullify_dots_bool
5098 { \phantom { \ensuremath { \@@_old_iddots } } }
5099 \bool_gset_true:N \g_@@_empty_cell_bool
5100 }
5101 }

```

End of the `\AddToHook`.

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

```

5102 \keys_define:nn { nicematrix / Ddots }
5103 {
5104     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5105     draw-first .default:n = true ,
5106     draw-first .value_forbidden:n = true
5107 }

```

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

```

5108 \cs_new_protected:Npn \@@_Hspace:
5109 {
5110     \bool_gset_true:N \g_@@_empty_cell_bool
5111     \hspace
5112 }

```

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`.

```

5113 \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`.

```

5114 \cs_new:Npn \@@_Hdotsfor:
5115 {
5116     \bool_lazy_and:nnTF
5117     { \int_if_zero_p:n \c@jCol }
5118     { \int_if_zero_p:n \l_@@_first_col_int }
5119     {
5120         \bool_if:NTF \g_@@_after_col_zero_bool
5121         {
5122             \multicolumn { 1 } { c } { }
5123             \@@_Hdotsfor_i
5124         }
5125         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5126     }
5127     {
5128         \multicolumn { 1 } { c } { }
5129         \@@_Hdotsfor_i
5130     }
5131 }

```

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:`).

```

5132 \hook_gput_code:nnn { begindocument } { . }
5133 {
5134     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5135     \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.

```

5136 \cs_new_protected:Npn \@@_Hdotsfor_i
5137 { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5138 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5139 {
5140   \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5141   {
5142     \@@_Hdotsfor:nnnn
5143     { \int_use:N \c@iRow }
5144     { \int_use:N \c@jCol }
5145     { #2 }
5146     {
5147       #1 , #3 ,
5148       down = \exp_not:n { #4 } ,
5149       up = \exp_not:n { #5 } ,
5150       middle = \exp_not:n { #6 }
5151     }
5152   }
5153   \prg_replicate:nn { #2 - 1 }
5154   {
5155     &
5156     \multicolumn { 1 } { c } { }
5157     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5158   }
5159 }
5160 }

```

```

5161 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5162 {
5163   \bool_set_false:N \l_@@_initial_open_bool
5164   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5165 \int_set:Nn \l_@@_initial_i_int { #1 }
5166 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5167 \int_compare:nNnTF { #2 } = \c_one_int
5168 {
5169   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5170   \bool_set_true:N \l_@@_initial_open_bool
5171 }
5172 {
5173   \cs_if_exist:cTF
5174   {
5175     pgf @ sh @ ns @ \@@_env:
5176     - \int_use:N \l_@@_initial_i_int
5177     - \int_eval:n { #2 - 1 }
5178   }
5179   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5180   {
5181     \int_set:Nn \l_@@_initial_j_int { #2 }
5182     \bool_set_true:N \l_@@_initial_open_bool
5183   }
5184 }
5185 \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5186 {
5187   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5188   \bool_set_true:N \l_@@_final_open_bool
5189 }
5190 {
5191   \cs_if_exist:cTF
5192   {

```

```

5193         pgf @ sh @ ns @ \@@_env:
5194         - \int_use:N \l_@@_final_i_int
5195         - \int_eval:n { #2 + #3 }
5196     }
5197     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5198     {
5199         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5200         \bool_set_true:N \l_@@_final_open_bool
5201     }
5202 }
5203 \group_begin:
5204 \@@_open_shorten:
5205 \int_if_zero:nTF { #1 }
5206 { \color { nicematrix-first-row } }
5207 {
5208     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5209     { \color { nicematrix-last-row } }
5210 }
5211
5212 \keys_set:nn { nicematrix / xdots } { #4 }
5213 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5214 \@@_actually_draw_Ldots:
5215 \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).

```

5216     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5217     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5218 }
5219 \hook_gput_code:nnn { begindocument } { . }
5220 {
5221     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5222     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5223     \cs_new_protected:Npn \@@_Vdotsfor:
5224     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5225     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5226     {
5227         \bool_gset_true:N \g_@@_empty_cell_bool
5228         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5229         {
5230             \@@_Vdotsfor:nnnn
5231             { \int_use:N \c@iRow }
5232             { \int_use:N \c@jCol }
5233             { #2 }
5234             {
5235                 #1 , #3 ,
5236                 down = \exp_not:n { #4 } ,
5237                 up = \exp_not:n { #5 } ,
5238                 middle = \exp_not:n { #6 }
5239             }
5240         }
5241     }
5242 }
5243 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5244 {
5245     \bool_set_false:N \l_@@_initial_open_bool
5246     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
5247 \int_set:Nn \l_@@_initial_j_int { #2 }
5248 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

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

```
5249 \int_compare:nNnTF { #1 } = \c_one_int
5250 {
5251   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5252   \bool_set_true:N \l_@@_initial_open_bool
5253 }
5254 {
5255   \cs_if_exist:cTF
5256   {
5257     pgf @ sh @ ns @ \@@_env:
5258     - \int_eval:n { #1 - 1 }
5259     - \int_use:N \l_@@_initial_j_int
5260   }
5261   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5262   {
5263     \int_set:Nn \l_@@_initial_i_int { #1 }
5264     \bool_set_true:N \l_@@_initial_open_bool
5265   }
5266 }
5267 \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5268 {
5269   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5270   \bool_set_true:N \l_@@_final_open_bool
5271 }
5272 {
5273   \cs_if_exist:cTF
5274   {
5275     pgf @ sh @ ns @ \@@_env:
5276     - \int_eval:n { #1 + #3 }
5277     - \int_use:N \l_@@_final_j_int
5278   }
5279   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5280   {
5281     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5282     \bool_set_true:N \l_@@_final_open_bool
5283   }
5284 }
5285 \group_begin:
5286 \@@_open_shorten:
5287 \int_if_zero:nTF { #2 }
5288 { \color { nicematrix-first-col } }
5289 {
5290   \int_compare:nNnT { #2 } = \g_@@_col_total_int
5291   { \color { nicematrix-last-col } }
5292 }
5293 \keys_set:nn { nicematrix / xdots } { #4 }
5294 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5295 \@@_actually_draw_Vdots:
5296 \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).

```
5297 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5298 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5299 }
```

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

```

5300 \NewDocumentCommand \@@_rotate: { 0 { } }
5301 {
5302   \peek_remove_spaces:n
5303   {
5304     \bool_gset_true:N \g_@@_rotate_bool
5305     \keys_set:nn { nicematrix / rotate } { #1 }
5306   }
5307 }

5308 \keys_define:nn { nicematrix / rotate }
5309 {
5310   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5311   c .value_forbidden:n = true ,
5312   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5313 }

```

20 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

```

5314 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5315 {
5316   \tl_if_empty:nTF { #2 }
5317   { #1 }
5318   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5319 }
5320 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5321 { \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).

```

5322 \hook_gput_code:nnn { begindocument } { . }
5323 {
5324   \cs_set_nopar:Npn \l_@@_argspec_tl
5325   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } { } } }
5326   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5327   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5328   {
5329     \group_begin:
5330     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5331     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5332     \use:e

```

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

```

5333         {
5334             \@@_line_i:nn
5335             { \@@_double_int_eval:n #2 - \q_stop }
5336             { \@@_double_int_eval:n #3 - \q_stop }
5337         }
5338     \group_end:
5339 }
5340 }

5341 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5342 {
5343     \bool_set_false:N \l_@@_initial_open_bool
5344     \bool_set_false:N \l_@@_final_open_bool
5345     \bool_lazy_or:nnTF
5346     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5347     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5348     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5349     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5350 }

5351 \hook_gput_code:nnn { begindocument } { . }
5352 {
5353     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5354     {

```

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:`.

```

5355     \c_@@_pgfortikzpicture_tl
5356     \@@_draw_line_iii:nn { #1 } { #2 }
5357     \c_@@_endpgfortikzpicture_tl
5358 }
5359 }

```

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

```

5360 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5361 {
5362     \pgfrememberpicturepositiononpagetrue
5363     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5364     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5365     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5366     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5367     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5368     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5369     \@@_draw_line:
5370 }

```

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).

21 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

```
5371 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5372 { \int_compare:nNtT { \c@iRow } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```
5373 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5374 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5375 {
5376   \tl_gput_right:Ne \g_@@_row_style_tl
5377   {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5378   \exp_not:N
5379   \@@_if_row_less_than:nn
5380   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
```

The `\scan_stop:` is mandatory (for ex. for the case where `\rotate` is used in the argument of `\RowStyle`).

```
5381   { \exp_not:n { #1 } \scan_stop: }
5382 }
5383 }
```

```
5384 \keys_define:nn { nicematrix / RowStyle }
5385 {
5386   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5387   cell-space-top-limit .value_required:n = true ,
5388   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5389   cell-space-bottom-limit .value_required:n = true ,
5390   cell-space-limits .meta:n =
5391   {
5392     cell-space-top-limit = #1 ,
5393     cell-space-bottom-limit = #1 ,
5394   } ,
5395   color .tl_set:N = \l_@@_color_tl ,
5396   color .value_required:n = true ,
5397   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5398   bold .default:n = true ,
5399   nb-rows .code:n =
5400   \str_if_eq:eeTF { #1 } { * }
5401   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5402   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5403   nb-rows .value_required:n = true ,
5404   rowcolor .tl_set:N = \l_tmpa_tl ,
5405   rowcolor .value_required:n = true ,
5406   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5407 }
```

```
5408 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5409 {
5410   \group_begin:
5411   \tl_clear:N \l_tmpa_tl
5412   \tl_clear:N \l_@@_color_tl
5413   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5414   \dim_zero:N \l_tmpa_dim
5415   \dim_zero:N \l_tmpb_dim
5416   \keys_set:nn { nicematrix / RowStyle } { #1 }
```

If the key `rowcolor` has been used.

```
5417 \tl_if_empty:NF \l_tmpa_tl
5418 {
```

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

```
5419 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5420 {
```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```
5421 \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5422 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5423 { \int_use:N \c@iRow - * }
5424 }
```

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

```
5425 \int_compare:nNt \l_@@_key_nb_rows_int > \c_one_int
5426 {
5427 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5428 {
5429 \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5430 {
5431 \int_eval:n { \c@iRow + 1 }
5432 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5433 }
5434 }
5435 }
5436 }
5437 \@@_put_in_row_style:n { \exp_not:n { #2 } }
```

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

```
5438 \dim_compare:nNt \l_tmpa_dim > \c_zero_dim
5439 {
5440 \@@_put_in_row_style:e
5441 {
5442 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5443 {
```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```
5444 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5445 { \dim_use:N \l_tmpa_dim }
5446 }
5447 }
5448 }
```

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

```
5449 \dim_compare:nNt \l_tmpb_dim > \c_zero_dim
5450 {
5451 \@@_put_in_row_style:e
5452 {
5453 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5454 {
5455 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5456 { \dim_use:N \l_tmpb_dim }
5457 }
5458 }
5459 }
```

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

```
5460 \tl_if_empty:NF \l_@@_color_tl
5461 {
5462 \@@_put_in_row_style:e
5463 {
5464 \mode_leave_vertical:
5465 \@@_color:n { \l_@@_color_tl }
5466 }
5467 }
```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5468   \bool_if:NT \l_@@_bold_row_style_bool
5469   {
5470     \@@_put_in_row_style:n
5471     {
5472       \exp_not:n
5473       {
5474         \if_mode_math:
5475           \c_math_toggle_token
5476           \bfseries \boldmath
5477           \c_math_toggle_token
5478         \else:
5479           \bfseries \boldmath
5480         \fi:
5481       }
5482     }
5483   }
5484   \group_end:
5485   \g_@@_row_style_tl
5486   \ignorespaces
5487 }

```

22 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).

```

5488 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5489 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5490 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5491 {

```

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.

```

5492   \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`. `\str_if_in:nnF` is mandatory: don't use `\tl_if_in:nnF`.

```

5493   \str_if_in:nnF { #1 } { !! }
5494   {
5495     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5496     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5497   }
5498   \int_if_zero:nTF \l_tmpa_int

```

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

```

5499   {
5500     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5501     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5502   }

```

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

```

5503   { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5504 }

```

The following command must be used within a `\pgfpicture`.

```

5505 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5506 {
5507   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5508   {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5509     \group_begin:
5510     \pgfsetcornersarced
5511     {
5512       \pgfpoint
5513       { \l_@@_tab_rounded_corners_dim }
5514       { \l_@@_tab_rounded_corners_dim }
5515     }

```

Because we want `nicematrix` compatible with arrays constructed by `array`, the nodes for the rows and columns (that is to say the nodes `row-i` and `col-j`) have not always the expected position, that is to say, there is sometimes a slight shifting of something such as `\arrayrulewidth`. Now, for the clipping, we have to change slightly the position of that clipping whether a rounded rectangle around the array is required. That's the point which is tested in the following line.

```

5516   \bool_if:NTF \l_@@_hvlines_bool
5517   {
5518     \pgfpathrectanglecorners
5519     {
5520       \pgfpointadd
5521       { \@@_qpoint:n { row-1 } }
5522       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5523     }
5524     {
5525       \pgfpointadd
5526       {
5527         \@@_qpoint:n
5528         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5529       }
5530       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5531     }
5532   }
5533   {
5534     \pgfpathrectanglecorners
5535     { \@@_qpoint:n { row-1 } }
5536     {
5537       \pgfpointadd
5538       {
5539         \@@_qpoint:n
5540         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5541       }
5542       { \pgfpoint \c_zero_dim \arrayrulewidth }
5543     }

```

```

5544     }
5545     \pgfusepath { clip }
5546     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5547     }
5548 }

```

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`).

```

5549 \cs_new_protected:Npn \@@_actually_color:
5550 {
5551   \pgfpicture
5552   \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.

```

5553   \@@_clip_with_rounded_corners:
5554   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5555   {
5556     \int_compare:nNnTF { ##1 } = \c_one_int
5557     {
5558       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5559       \use:c { g_@@_color _ 1 _tl }
5560       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5561     }
5562     {
5563       \begin { pgfscope }
5564         \@@_color_opacity ##2
5565         \use:c { g_@@_color _ ##1 _tl }
5566         \tl_gclear:c { g_@@_color _ ##1 _tl }
5567         \pgfusepath { fill }
5568       \end { pgfscope }
5569     }
5570   }
5571   \endpgfpicture
5572 }

```

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

```

5573 \cs_new_protected:Npn \@@_color_opacity
5574 {
5575   \peek_meaning:NTF [
5576     { \@@_color_opacity:w }
5577     { \@@_color_opacity:w [ ] }
5578   }

```

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.

```

5579 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5580 {
5581   \tl_clear:N \l_tmpa_tl
5582   \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.

```

5583   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5584   \tl_if_empty:NTF \l_tmpb_tl
5585   { \@declaredcolor }
5586   { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5587 }

```

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

```

5588 \keys_define:nn { nicematrix / color-opacity }
5589 {
5590   opacity .tl_set:N          = \l_tmpa_tl ,
5591   opacity .value_required:n = true
5592 }

5593 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5594 {
5595   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5596   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5597   \@@_cartesian_path:
5598 }

```

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

```

5599 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5600 {
5601   \tl_if_blank:nF { #2 }
5602   {
5603     \@@_add_to_colors_seq:en
5604     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5605     { \@@_cartesian_color:nn { #3 } { - } }
5606   }
5607 }

```

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

```

5608 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5609 {
5610   \tl_if_blank:nF { #2 }
5611   {
5612     \@@_add_to_colors_seq:en
5613     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5614     { \@@_cartesian_color:nn { - } { #3 } }
5615   }
5616 }

```

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

```

5617 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5618 {
5619   \tl_if_blank:nF { #2 }
5620   {
5621     \@@_add_to_colors_seq:en
5622     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5623     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5624   }
5625 }

```

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

```

5626 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5627 {
5628   \tl_if_blank:nF { #2 }
5629   {
5630     \@@_add_to_colors_seq:en
5631     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5632     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5633   }
5634 }

```

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

```

5635 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5636 {
5637   \@@_cut_on_hyphen:w #1 \q_stop
5638   \tl_clear_new:N \l_@@_tmpc_tl
5639   \tl_clear_new:N \l_@@_tmpd_tl
5640   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5641   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5642   \@@_cut_on_hyphen:w #2 \q_stop
5643   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5644   \tl_set:Ne \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`.

```

5645   \@@_cartesian_path:n { #3 }
5646 }

```

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

```

5647 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5648 {
5649   \clist_map_inline:nn { #3 }
5650     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5651 }

```

```

5652 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5653 {
5654   \int_step_inline:nn \c@iRow
5655   {
5656     \int_step_inline:nn \c@jCol
5657     {
5658       \int_if_even:nTF { #####1 + ##1 }
5659         { \@@_cellcolor [ #1 ] { #2 } }
5660         { \@@_cellcolor [ #1 ] { #3 } }
5661       { ##1 - #####1 }
5662     }
5663   }
5664 }

```

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”.

```

5665 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5666 {
5667   \@@_rectanglecolor [ #1 ] { #2 }
5668   { 1 - 1 }
5669   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5670 }

5671 \keys_define:nn { nicematrix / rowcolors }
5672 {
5673   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5674   respect-blocks .default:n = true ,
5675   cols .tl_set:N = \l_@@_cols_tl ,
5676   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5677   restart .default:n = true ,
5678   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5679 }

```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`.

Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

In `nicematrix`, the command `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`.

#1 (optional) is the color space; **#2** is a list of intervals of rows; **#3** is the list of colors; **#4** is for the optional list of pairs *key=value*.

```
5680 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5681 {
```

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

```
5682 \group_begin:
5683 \seq_clear_new:N \l_@@_colors_seq
5684 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5685 \tl_clear_new:N \l_@@_cols_tl
5686 \cs_set_nopar:Npn \l_@@_cols_tl { - }
5687 \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).

```
5688 \int_zero_new:N \l_@@_color_int
5689 \int_set_eq:NN \l_@@_color_int \c_one_int
5690 \bool_if:NT \l_@@_respect_blocks_bool
5691 {
```

We don't want to take into account a block which is completely in the “first column” (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```
5692 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5693 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5694 { \@@_not_in_exterior_p:nnnnn ##1 }
5695 }
5696 \pgfpicture
5697 \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5698 \clist_map_inline:nn { #2 }
5699 {
5700 \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5701 \tl_if_in:NnTF \l_tmpa_tl { - }
5702 { \@@_cut_on_hyphen:w ##1 \q_stop }
5703 { \tl_set:No \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.

```
5704 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5705 \int_set:Nn \l_@@_color_int
5706 { \bool_if:NNTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5707 \int_zero_new:N \l_@@_tmpc_int
5708 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5709 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5710 {
```

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

```
5711 \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).

```
5712 \bool_if:NT \l_@@_respect_blocks_bool
5713 {
5714 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5715 { \@@_intersect_our_row_p:nnnnn ###1 }
5716 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }
```

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

```
5717 }
5718 \tl_set:No \l_@@_rows_tl
5719 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```


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

```

5720         \tl_clear_new:N \l_@@_color_tl
5721         \tl_set:Nx \l_@@_color_tl
5722         {
5723             \@@_color_index:n
5724             {
5725                 \int_mod:nn
5726                 { \l_@@_color_int - 1 }
5727                 { \seq_count:N \l_@@_colors_seq }
5728                 + 1
5729             }
5730         }
5731         \tl_if_empty:NF \l_@@_color_tl
5732         {
5733             \@@_add_to_colors_seq:ee
5734             { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5735             { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5736         }
5737         \int_incr:N \l_@@_color_int
5738         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5739     }
5740 }
5741 \endpgfpicture
5742 \group_end:
5743 }

```

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.

```

5744 \cs_new:Npn \@@_color_index:n #1
5745 {
5746     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5747     { \@@_color_index:n { #1 - 1 } }
5748     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5749 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5750 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5751 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5752 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5753 {
5754     \int_compare:nNtT { #3 } > \l_tmpb_int
5755     { \int_set:Nn \l_tmpb_int { #3 } }
5756 }

5757 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5758 {
5759     \int_if_zero:nTF { #4 }
5760     \prg_return_false:
5761     {
5762         \int_compare:nNtTF { #2 } > \c@jCol
5763         \prg_return_false:
5764         \prg_return_true:
5765     }
5766 }

```

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

```

5767 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5768 {
5769   \int_compare:nNnTF { #1 } > \l_tmpa_int
5770     \prg_return_false:
5771     {
5772       \int_compare:nNnTF \l_tmpa_int > { #3 }
5773       \prg_return_false:
5774       \prg_return_true:
5775     }
5776 }

```

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).

```

5777 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5778 {
5779   \dim_compare:nNnTF { #1 } = \c_zero_dim
5780   {
5781     \bool_if:NTF
5782       \l_@@_nocolor_used_bool
5783       \@@_cartesian_path_normal_ii:
5784       {
5785         \clist_if_empty:NTF \l_@@_corners_cells_clist
5786         { \@@_cartesian_path_normal_i:n { #1 } }
5787         \@@_cartesian_path_normal_ii:
5788       }
5789     }
5790   { \@@_cartesian_path_normal_i:n { #1 } }
5791 }

```

First, the situation where is a rectangular zone of cells will be colored as a whole (in the instructions of the resulting PDF). The argument is the radius of the corners.

```

5792 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5793 {
5794   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5795 \clist_map_inline:Nn \l_@@_cols_tl
5796 {
5797   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5798   \tl_if_in:NnTF \l_tmpa_tl { - }
5799   { \@@_cut_on_hyphen:w ##1 \q_stop }
5800   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5801   \tl_if_empty:NTF \l_tmpa_tl
5802   { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5803   {
5804     \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5805     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5806   }
5807   \tl_if_empty:NTF \l_tmpb_tl
5808   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5809   {
5810     \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5811     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5812   }
5813   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5814   { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5815 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5816 \@@_qpoint:n { col - \l_tmpa_tl }
5817 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5818 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5819 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5820 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5821 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5822 \clist_map_inline:Nn \l_@@_rows_tl
5823 {
5824   \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5825   \tl_if_in:NnTF \l_tmpa_tl { - }
5826   { \@@_cut_on_hyphen:w ####1 \q_stop }
5827   { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5828   \tl_if_empty:NnTF \l_tmpa_tl
5829   { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5830   {
5831     \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5832     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5833   }
5834   \tl_if_empty:NnTF \l_tmpb_tl
5835   { \tl_set:Nn \l_tmpb_tl { \int_use:N \c@iRow } }
5836   {
5837     \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5838     { \tl_set:Nn \l_tmpb_tl { \int_use:N \c@iRow } }
5839   }
5840   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5841   { \tl_set:Nn \l_tmpb_tl { \int_use:N \g_@@_row_total_int } }

```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```

5842 \cs_if_exist:cF
5843 { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5844 {
5845   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5846   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5847   \@@_qpoint:n { row - \l_tmpa_tl }
5848   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5849   \pgfpathrectanglecorners
5850   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5851   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5852 }
5853 }
5854 }
5855 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5856 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5857 {
5858   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5859   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5860 \clist_map_inline:Nn \l_@@_cols_tl
5861 {
5862   \@@_qpoint:n { col - ##1 }
5863   \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5864   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5865   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5866   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5867   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5868 \clist_map_inline:Nn \l_@@_rows_tl
5869 {
5870   \@@_clist_if_in:NnF \l_@@_corners_cells_clist
5871   { #####1 - ##1 }
5872   {
5873     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5874     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5875     \@@_qpoint:n { row - #####1 }
5876     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5877     \cs_if_exist:cF { @@ - #####1 - ##1 - nocolor }
5878     {
5879       \pgfpathrectanglecorners
5880       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5881       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5882     }
5883   }
5884 }
5885 }
5886 }

```

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).

```

5887 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n \c_zero_dim }

```

Despite its name, the following command does not create a PGF path. It declares as colored by the “empty color” all the cells in what would be the path. Hence, the other coloring instructions of nicematrix won’t put color in those cells. the

```

5888 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5889 {
5890   \bool_set_true:N \l_@@_nocolor_used_bool
5891   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5892   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5893 \clist_map_inline:Nn \l_@@_rows_tl
5894 {
5895   \clist_map_inline:Nn \l_@@_cols_tl
5896   { \cs_set:cpn { @@ - ##1 - #####1 - nocolor } { } }
5897 }
5898 }

```

The following command will be used only with \l_@@_cols_tl and \c@jCol (first case) or with \l_@@_rows_tl and \c@iRow (second case). For instance, with \l_@@_cols_tl equal to 2,4-6,8-* and \c@jCol equal to 10, the clist \l_@@_cols_tl will be replaced by 2,4,5,6,8,9,10.

```

5899 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5900 {
5901   \clist_set_eq:NN \l_tmpa_clist #1
5902   \clist_clear:N #1
5903   \clist_map_inline:Nn \l_tmpa_clist
5904   {
5905     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5906     \tl_if_in:NnTF \l_tmpa_tl { - }
5907     { \@@_cut_on_hyphen:w ##1 \q_stop }
5908     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5909     \bool_lazy_or:nnT
5910     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5911     { \tl_if_blank_p:o \l_tmpa_tl }
5912     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5913     \bool_lazy_or:nnT
5914     { \str_if_eq_p:ee \l_tmpb_tl { * } }

```

```

5915         { \tl_if_blank_p:o \l_tmpb_tl }
5916         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5917     \int_compare:nNnT \l_tmpb_tl > #2
5918         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5919     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5920         { \clist_put_right:Nn #1 { #####1 } }
5921 }
5922 }

```

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

```

5923 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5924 {
5925     \@@_test_color_inside:
5926     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5927     {

```

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`).

```

5928         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5929         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5930     }
5931     \ignorespaces
5932 }

```

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

```

5933 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5934 {
5935     \@@_test_color_inside:
5936     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5937     {
5938         \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5939         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5940         { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5941     }
5942     \ignorespaces
5943 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5944 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5945 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5946 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5947 {
5948     \@@_test_color_inside:
5949     \peek_remove_spaces:n
5950     { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5951 }

5952 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5953 {

```

A use of `\rowlistcolors` in the tabular erases the instructions `\rowlistcolors` which are in force. However, it's possible to put *several* instructions `\rowlistcolors` in the same row of a tabular: it may be useful when those instructions `\rowlistcolors` concerns different columns of the tabular (thanks to the key `cols` of `\rowlistcolors`). That's why we store the different instructions `\rowlistcolors` which are in force in a sequence `\g_@@_rowlistcolors_seq`. Now, we will filter that sequence to keep only the elements which have been issued on the actual row. We will store the elements to keep in the `\g_tmpa_seq`.

```

5954 \seq_gclear:N \g_tmpa_seq
5955 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5956 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5957 \seq_gset_eq:NN \g_@@_rowlistcolors_seq \g_tmpa_seq

```

Now, we add to the sequence `\g_@@_rowlistcolors_seq` (which is the list of the commands `\rowlistcolors` which are in force) the current instruction `\rowlistcolors`.

```

5958 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5959 {
5960   { \int_use:N \c@iRow }
5961   { \exp_not:n { #1 } }
5962   { \exp_not:n { #2 } }
5963   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5964 }
5965 }

```

The following command will be applied to each component of `\g_@@_rowlistcolors_seq`. Each component of that sequence is a kind of 4-uple of the form `{#1}{#2}{#3}{#4}`.

#1 is the number of the row where the command `\rowlistcolors` has been issued.

#2 is the colorimetric space (optional argument of the `\rowlistcolors`).

#3 is the list of colors (mandatory argument of `\rowlistcolors`).

#4 is the list of *key=value* pairs (last optional argument of `\rowlistcolors`).

```

5966 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5967 {
5968   \int_compare:nNnTF { #1 } = \c@iRow

```

We (temporary) keep in memory in `\g_tmpa_seq` the instructions which will still be in force after the current instruction (because they have been issued in the same row of the tabular).

```

5969   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5970   {
5971     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5972     {
5973       \@@_rowlistcolors
5974       [ \exp_not:n { #2 } ]
5975       { #1 - \int_eval:n { \c@iRow - 1 } }
5976       { \exp_not:n { #3 } }
5977       [ \exp_not:n { #4 } ]
5978     }
5979   }
5980 }

```

The following command will be used at the end of the tabular, just before the execution of the `\g_@@_pre_code_before_tl`. It clears the sequence `\g_@@_rowlistcolors_seq` of all the commands `\rowlistcolors` which are (still) in force.

```

5981 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5982 {
5983   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5984   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5985   \seq_gclear:N \g_@@_rowlistcolors_seq
5986 }

```

```

5987 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5988 {
5989   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5990   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5991 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the `pre-\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5992 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5993 {

```

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).

```

5994   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5995   {

```

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).

```

5996     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5997     {
5998       \exp_not:N \columncolor [ #1 ]
5999       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6000     }
6001   }
6002 }

6003 \hook_gput_code:nnn { begindocument } { . }
6004 {
6005   \IfPackageLoadedTF { colortbl }
6006   {
6007     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
6008     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
6009     \cs_new_protected:Npn \@@_revert_colortbl:
6010     {
6011       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
6012       {
6013         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
6014         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
6015       }
6016     }
6017   }
6018   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
6019 }

```

23 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
6020 \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`.

```
6021 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6022 {
6023   \int_if_zero:nTF \l_@@_first_col_int
6024   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6025   {
6026     \int_if_zero:nTF \c@jCol
6027     {
6028       \int_compare:nNnF \c@iRow = { -1 }
6029       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6030     }
6031     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6032   }
6033 }
```

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.

```
6034 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6035 {
6036   \int_if_zero:nF \c@iRow
6037   {
6038     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6039     {
6040       \int_compare:nNnT \c@jCol > \c_zero_int
6041       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6042     }
6043   }
6044 }
```

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.

```
6045 \keys_define:nn { nicematrix / Rules }
6046 {
6047   position .int_set:N = \l_@@_position_int ,
6048   position .value_required:n = true ,
6049   start .int_set:N = \l_@@_start_int ,
6050   end .code:n =
6051     \bool_lazy_or:nnTF
6052     { \tl_if_empty_p:n { #1 } }
6053     { \str_if_eq_p:ee { #1 } { last } }
6054     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6055     { \int_set:Nn \l_@@_end_int { #1 } }
6056 }
```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous

rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6057 \keys_define:nn { nicematrix / RulesBis }
6058 {
6059   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6060   multiplicity .initial:n = 1 ,
6061   dotted .bool_set:N = \l_@@_dotted_bool ,
6062   dotted .initial:n = false ,
6063   dotted .default:n = true ,

```

We want that, even when the rule has been defined with TikZ by the key `tikz`, the user has still the possibility to change the color of the rule with the key `color` (in the command `\Hline`, not in the key `tikz` of the command `\Hline`). The main use is, when the user has defined its own command `\MyDashedLine` by `\newcommand{\MyDashedRule}{\Hline[tikz=dashed]}`, to give the ability to write `\MyDashedRule[color=red]`.

```

6064   color .code:n =
6065     \@@_set_CT@arc@:n { #1 }
6066     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6067   color .value_required:n = true ,
6068   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6069   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.

```

6070   tikz .code:n =
6071     \IfPackageLoadedTF { tikz }
6072       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6073       { \@@_error:n { tikz~without~tikz } } ,
6074   tikz .value_required:n = true ,
6075   total-width .dim_set:N = \l_@@_rule_width_dim ,
6076   total-width .value_required:n = true ,
6077   width .meta:n = { total-width = #1 } ,
6078   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
6079 }

```

The vertical rules

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

```

6080 \cs_new_protected:Npn \@@_vline:n #1
6081 {

```

The group is for the options.

```

6082   \group_begin:
6083   \int_set_eq:NN \l_@@_end_int \c@iRow
6084   \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).

```

6085   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6086     \@@_vline_i:
6087   \group_end:
6088 }
6089 \cs_new_protected:Npn \@@_vline_i:
6090 {

```

`\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`.

```

6091   \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6092   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6093     \l_tmpa_tl
6094   {

```

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.

```

6095     \bool_gset_true:N \g_tmpa_bool
6096     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6097       { \@@_test_vline_in_block:nnnnn ##1 }
6098     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6099       { \@@_test_vline_in_block:nnnnn ##1 }
6100     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6101       { \@@_test_vline_in_stroken_block:nnnn ##1 }
6102     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6103     \bool_if:NTF \g_tmpa_bool
6104       {
6105         \int_if_zero:nT \l_@@_local_start_int

```

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.

```

6106         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6107       }
6108     {
6109       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6110       {
6111         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6112         \@@_vline_ii:
6113         \int_zero:N \l_@@_local_start_int
6114       }
6115     }
6116   }
6117   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6118   {
6119     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6120     \@@_vline_ii:
6121   }
6122 }

```

```

6123 \cs_new_protected:Npn \@@_test_in_corner_v:
6124 {
6125   \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6126   {
6127     \@@_clist_if_in:NeT
6128     \l_@@_corners_cells_clist
6129     { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6130     { \bool_set_false:N \g_tmpa_bool }
6131   }
6132   {
6133     \@@_clist_if_in:NeT
6134     \l_@@_corners_cells_clist
6135     { \l_tmpa_tl - \l_tmpb_tl }
6136     {
6137       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6138       { \bool_set_false:N \g_tmpa_bool }
6139       {
6140         \@@_clist_if_in:NeT
6141         \l_@@_corners_cells_clist
6142         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6143         { \bool_set_false:N \g_tmpa_bool }
6144       }
6145     }
6146   }
6147 }

```

```

6148 \cs_new_protected:Npn \@@_vline_ii:
6149 {
6150   \tl_clear:N \l_@@_tikz_rule_tl
6151   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6152   \bool_if:NTF \l_@@_dotted_bool
6153     \@@_vline_iv:
6154     {
6155       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6156         \@@_vline_iii:
6157         \@@_vline_v:
6158     }
6159 }

```

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

```

6160 \cs_new_protected:Npn \@@_vline_iii:
6161 {
6162   \pgfpicture
6163   \pgfrememberpicturepositiononpagetrue
6164   \pgf@relevantforpicturesizefalse
6165   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6166   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6167   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6168   \dim_set:Nn \l_tmpb_dim
6169   {
6170     \pgf@x
6171     - 0.5 \l_@@_rule_width_dim
6172     +
6173     ( \arrayrulewidth * \l_@@_multiplicity_int
6174       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6175   }
6176   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6177   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6178   \bool_lazy_all:nT
6179   {
6180     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6181     { \cs_if_exist_p:N \CT@drsc@ }
6182     { ! \tl_if_blank_p:o \CT@drsc@ }
6183   }
6184   {
6185     \group_begin:
6186     \CT@drsc@
6187     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6188     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6189     \dim_set:Nn \l_@@_tmpd_dim
6190     {
6191       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6192       * ( \l_@@_multiplicity_int - 1 )
6193     }
6194     \pgfpathrectanglecorners
6195     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6196     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6197     \pgfusepath { fill }
6198     \group_end:
6199   }
6200   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6201   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6202   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6203   {
6204     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6205     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6206     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6207     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6208   }

```

```

6209 \CT@arc@
6210 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6211 \pgfsetrectcap
6212 \pgfusepathqstroke
6213 \endpgfpicture
6214 }

```

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

```

6215 \cs_new_protected:Npn \@@_vline_iv:
6216 {
6217   \pgfpicture
6218   \pgfrememberpicturepositiononpagetrue
6219   \pgf@relevantforpicturesizefalse
6220   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6221   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6222   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6223   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6224   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6225   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6226   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6227   \CT@arc@
6228   \@@_draw_line:
6229   \endpgfpicture
6230 }

```

The following code is for the case when the user uses the key `tikz`.

```

6231 \cs_new_protected:Npn \@@_vline_v:
6232 {
6233   \begin {tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6234 \CT@arc@
6235 \tl_if_empty:NF \l_@@_rule_color_tl
6236 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6237 \pgfrememberpicturepositiononpagetrue
6238 \pgf@relevantforpicturesizefalse
6239 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6240 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6241 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6242 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6243 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6244 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6245 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6246 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6247 ( \l_tmpb_dim , \l_tmpa_dim ) --
6248 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6249 \end { tikzpicture }
6250 }

```

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).

```

6251 \cs_new_protected:Npn \@@_draw_vlines:
6252 {
6253   \int_step_inline:nnn
6254   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6255   {
6256     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6257     \c@jCol
6258     { \int_eval:n { \c@jCol + 1 } }
6259   }

```

```

6260 {
6261   \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6262   { \@@_clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6263   { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6264 }
6265 }

```

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}`.

```

6266 \cs_new_protected:Npn \@@_hline:n #1
6267 {

```

The group is for the options.

```

6268   \group_begin:
6269   \int_zero_new:N \l_@@_end_int
6270   \int_set_eq:NN \l_@@_end_int \c@jCol
6271   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6272   \@@_hline_i:
6273   \group_end:
6274 }

6275 \cs_new_protected:Npn \@@_hline_i:
6276 {
6277   \int_zero_new:N \l_@@_local_start_int
6278   \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`.

```

6279   \tl_set:N \l_tmpa_tl { \int_use:N \l_@@_position_int }
6280   \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6281   \l_tmpb_tl
6282   {

```

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.

```

6283     \bool_gset_true:N \g_tmpa_bool
6284     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6285     { \@@_test_hline_in_block:nnnnn ##1 }
6286     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6287     { \@@_test_hline_in_block:nnnnn ##1 }
6288     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6289     { \@@_test_hline_in_stroken_block:nnnn ##1 }
6290     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6291     \bool_if:NTF \g_tmpa_bool
6292     {
6293       \int_if_zero:nT \l_@@_local_start_int

```

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.

```

6294       { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6295     }
6296   {
6297     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6298     {
6299       \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6300       \@@_hline_ii:
6301       \int_zero:N \l_@@_local_start_int
6302     }
6303   }
6304 }

```

```

6305 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6306 {
6307     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6308     \@@_hline_ii:
6309 }
6310 }

6311 \cs_new_protected:Npn \@@_test_in_corner_h:
6312 {
6313     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6314     {
6315         \@@_clist_if_in:NeT
6316         \l_@@_corners_cells_clist
6317         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6318         { \bool_set_false:N \g_tmpa_bool }
6319     }
6320     {
6321         \@@_clist_if_in:NeT
6322         \l_@@_corners_cells_clist
6323         { \l_tmpa_tl - \l_tmpb_tl }
6324         {
6325             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6326             { \bool_set_false:N \g_tmpa_bool }
6327             {
6328                 \@@_clist_if_in:NeT
6329                 \l_@@_corners_cells_clist
6330                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6331                 { \bool_set_false:N \g_tmpa_bool }
6332             }
6333         }
6334     }
6335 }

6336 \cs_new_protected:Npn \@@_hline_ii:
6337 {
6338     \tl_clear:N \l_@@_tikz_rule_tl
6339     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6340     \bool_if:NTF \l_@@_dotted_bool
6341     \@@_hline_iv:
6342     {
6343         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6344         \@@_hline_iii:
6345         \@@_hline_v:
6346     }
6347 }

```

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

```

6348 \cs_new_protected:Npn \@@_hline_iii:
6349 {
6350     \pgfpicture
6351     \pgfrememberpicturepositiononpagetrue
6352     \pgf@relevantforpicturesizefalse
6353     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6354     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6355     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6356     \dim_set:Nn \l_tmpb_dim
6357     {
6358         \pgf@y
6359         - 0.5 \l_@@_rule_width_dim
6360         +
6361         ( \arrayrulewidth * \l_@@_multiplicity_int

```

```

6362         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6363     }
6364     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6365     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6366     \bool_lazy_all:nT
6367     {
6368         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6369         { \cs_if_exist_p:N \CT@drsc@ }
6370         { ! \tl_if_blank_p:o \CT@drsc@ }
6371     }
6372     {
6373         \group_begin:
6374         \CT@drsc@
6375         \dim_set:Nn \l_@@_tmpd_dim
6376         {
6377             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6378             * ( \l_@@_multiplicity_int - 1 )
6379         }
6380         \pgfpathrectanglecorners
6381         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6382         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6383         \pgfusepathqfill
6384         \group_end:
6385     }
6386     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6387     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6388     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6389     {
6390         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6391         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6392         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6393         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6394     }
6395     \CT@arc@
6396     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6397     \pgfsetrectcap
6398     \pgfusepathqstroke
6399     \endpgfpicture
6400 }

```

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}

```

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

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}

```

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

```

6401 \cs_new_protected:Npn \@@_hline_iv:
6402 {
6403     \pgfpicture
6404     \pgfrememberpicturepositiononpagetrue

```

```

6405 \pgf@relevantforpicturesizefalse
6406 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6407 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6408 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6409 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6410 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6411 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6412 {
6413   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6414   \bool_if:NF \g_@@_delims_bool
6415   { \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.

```

6416   \tl_if_eq:NnF \g_@@_left_delim_tl (
6417     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6418   )
6419 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6420 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6421 \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6422 {
6423   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6424   \bool_if:NF \g_@@_delims_bool
6425   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6426   \tl_if_eq:NnF \g_@@_right_delim_tl )
6427   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6428 }
6429 \CT@arc@
6430 \@@_draw_line:
6431 \endpgfpicture
6432 }

```

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`).

```

6433 \cs_new_protected:Npn \@@_hline_v:
6434 {
6435   \begin { tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6436 \CT@arc@
6437 \tl_if_empty:NF \l_@@_rule_color_tl
6438 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6439 \pgfrememberpicturepositiononpagetrue
6440 \pgf@relevantforpicturesizefalse
6441 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6442 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6443 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6444 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6445 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6446 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6447 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6448 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6449 ( \l_tmpa_dim , \l_tmpb_dim ) --
6450 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6451 \end { tikzpicture }
6452 }

```

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).


```

6453 \cs_new_protected:Npn \@@_draw_hlines:
6454 {
6455   \int_step_inline:nnn
6456     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6457   {
6458     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6459     \c@iRow
6460     { \int_eval:n { \c@iRow + 1 } }
6461   }
6462   {
6463     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6464     { \@@_clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6465     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6466   }
6467 }

```

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

```

6468 \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.

```

6469 \cs_set:Npn \@@_Hline_i:n #1
6470 {
6471   \peek_remove_spaces:n
6472   {
6473     \peek_meaning:NTF \Hline
6474     { \@@_Hline_ii:nn { #1 + 1 } }
6475     { \@@_Hline_iii:n { #1 } }
6476   }
6477 }
6478 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6479 \cs_set:Npn \@@_Hline_iii:n #1
6480 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6481 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6482 {
6483   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6484   \skip_vertical:N \l_@@_rule_width_dim
6485   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6486   {
6487     \@@_hline:n
6488     {
6489       multiplicity = #1 ,
6490       position = \int_eval:n { \c@iRow + 1 } ,
6491       total-width = \dim_use:N \l_@@_rule_width_dim ,
6492       #2
6493     }
6494   }
6495   \egroup
6496 }

```

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.

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```

6497 \cs_new_protected:Npn \@@_custom_line:n #1
6498 {
6499   \str_clear_new:N \l_@@_command_str
6500   \str_clear_new:N \l_@@_ccommand_str
6501   \str_clear_new:N \l_@@_letter_str
6502   \tl_clear_new:N \l_@@_other_keys_tl
6503   \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.

```

6504 \bool_lazy_all:nTF
6505 {
6506   { \str_if_empty_p:N \l_@@_letter_str }
6507   { \str_if_empty_p:N \l_@@_command_str }
6508   { \str_if_empty_p:N \l_@@_ccommand_str }
6509 }
6510 { \@@_error:n { No~letter~and~no~command } }
6511 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6512 }
6513 \keys_define:nn { nicematrix / custom-line }
6514 {
6515   letter .str_set:N = \l_@@_letter_str ,
6516   letter .value_required:n = true ,
6517   command .str_set:N = \l_@@_command_str ,
6518   command .value_required:n = true ,
6519   ccommand .str_set:N = \l_@@_ccommand_str ,
6520   ccommand .value_required:n = true ,
6521 }

```

```

6522 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6523 \cs_new_protected:Npn \@@_custom_line_i:n #1
6524 {

```

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

```

6525 \bool_set_false:N \l_@@_tikz_rule_bool
6526 \bool_set_false:N \l_@@_dotted_rule_bool
6527 \bool_set_false:N \l_@@_color_bool
6528 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6529 \bool_if:NT \l_@@_tikz_rule_bool
6530 {
6531   \IfPackageLoadedF { tikz }
6532   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6533   \bool_if:NT \l_@@_color_bool
6534   { \@@_error:n { color-in-custom-line-with-tikz } }
6535 }
6536 \bool_if:NT \l_@@_dotted_rule_bool
6537 {
6538   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6539   { \@@_error:n { key-multiplicity-with-dotted } }
6540 }
6541 \str_if_empty:NF \l_@@_letter_str
6542 {
6543   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6544   { \@@_error:n { Several~letters } }
6545   {
6546     \tl_if_in:NoTF
6547     \c_@@_forbidden_letters_str
6548     \l_@@_letter_str
6549     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6550   }

```

During the analyse of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6551 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6552 { \@@_v_custom_line:n { #1 } }

```

```

6553     }
6554   }
6555 }
6556 \str_if_empty:NF \l_@@_command_str { \l_@@_h_custom_line:n { #1 } }
6557 \str_if_empty:NF \l_@@_ccommand_str { \l_@@_c_custom_line:n { #1 } }
6558 }
6559 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6560 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\l_@@_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.

```

6561 \keys_define:nn { nicematrix / custom-line-bis }
6562 {
6563   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6564   multiplicity .initial:n = 1 ,
6565   multiplicity .value_required:n = true ,
6566   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6567   color .value_required:n = true ,
6568   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6569   tikz .value_required:n = true ,
6570   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6571   dotted .value_forbidden:n = true ,
6572   total-width .code:n = { } ,
6573   total-width .value_required:n = true ,
6574   width .code:n = { } ,
6575   width .value_required:n = true ,
6576   sep-color .code:n = { } ,
6577   sep-color .value_required:n = true ,
6578   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6579 }

```

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

```

6580 \bool_new:N \l_@@_dotted_rule_bool
6581 \bool_new:N \l_@@_tikz_rule_bool
6582 \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`.

```

6583 \keys_define:nn { nicematrix / custom-line-width }
6584 {
6585   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6586   multiplicity .initial:n = 1 ,
6587   multiplicity .value_required:n = true ,
6588   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6589   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6590                       \bool_set_true:N \l_@@_total_width_bool ,
6591   total-width .value_required:n = true ,
6592   width .meta:n = { total-width = #1 } ,
6593   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6594 }

```

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 `\l_@@_hline:n` (which is in the internal `\CodeAfter`).

```

6595 \cs_new_protected:Npn \l_@@_h_custom_line:n #1
6596 {

```

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` (which is in `\Hline`).

```

6597 \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6598 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6599 }

```

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`).

```

6600 \cs_new_protected:Npn \@@_c_custom_line:n #1
6601 {

```

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

```

6602 \exp_args:Nc \NewExpandableDocumentCommand
6603 { nicematrix - \l_@@_ccommand_str }
6604 { 0 { } m }
6605 {
6606 \noalign
6607 {
6608 \@@_compute_rule_width:n { #1 , ##1 }
6609 \skip_vertical:n { \l_@@_rule_width_dim }
6610 \clist_map_inline:nn
6611 { ##2 }
6612 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6613 }
6614 }
6615 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6616 }

```

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*.

```

6617 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6618 {
6619 \tl_if_in:nnTF { #2 } { - }
6620 { \@@_cut_on_hyphen:w #2 \q_stop }
6621 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6622 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6623 {
6624 \@@_hline:n
6625 {
6626 #1 ,
6627 start = \l_tmpa_tl ,
6628 end = \l_tmpb_tl ,
6629 position = \int_eval:n { \c@iRow + 1 } ,
6630 total-width = \dim_use:N \l_@@_rule_width_dim
6631 }
6632 }
6633 }

6634 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6635 {
6636 \bool_set_false:N \l_@@_tikz_rule_bool
6637 \bool_set_false:N \l_@@_total_width_bool
6638 \bool_set_false:N \l_@@_dotted_rule_bool
6639 \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6640 \bool_if:NF \l_@@_total_width_bool
6641 {
6642 \bool_if:NTF \l_@@_dotted_rule_bool
6643 { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6644 {
6645 \bool_if:NF \l_@@_tikz_rule_bool
6646 {

```

```

6647         \dim_set:Nn \l_@@_rule_width_dim
6648         {
6649             \arrayrulewidth * \l_@@_multiplicity_int
6650             + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6651         }
6652     }
6653 }
6654 }
6655 }
6656 \cs_new_protected:Npn \@@_v_custom_line:n #1
6657 {
6658     \@@_compute_rule_width:n { #1 }

```

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

```

6659     \tl_gput_right:Ne \g_@@_array_preamble_tl
6660     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6661     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6662     {
6663         \@@_vline:n
6664         {
6665             #1 ,
6666             position = \int_eval:n { \c@jCol + 1 } ,
6667             total-width = \dim_use:N \l_@@_rule_width_dim
6668         }
6669     }
6670     \@@_rec_preamble:n
6671 }
6672 \@@_custom_line:n
6673 { 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.

```

6674 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6675 {
6676     \int_compare:nNnT \l_tmpa_tl > { #1 }
6677     {
6678         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6679         {
6680             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6681             {
6682                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6683                 { \bool_gset_false:N \g_tmpa_bool }
6684             }
6685         }
6686     }
6687 }

```

The same for vertical rules.

```

6688 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6689 {
6690     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6691     {
6692         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6693         {
6694             \int_compare:nNnT \l_tmpb_tl > { #2 }
6695             {
6696                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6697                 { \bool_gset_false:N \g_tmpa_bool }
6698             }
6699         }
6700     }

```

```

6699     }
6700   }
6701 }
6702 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6703 {
6704   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6705   {
6706     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6707     {
6708       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6709       { \bool_gset_false:N \g_tmpa_bool }
6710       {
6711         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6712         { \bool_gset_false:N \g_tmpa_bool }
6713       }
6714     }
6715   }
6716 }
6717 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6718 {
6719   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6720   {
6721     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6722     {
6723       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6724       { \bool_gset_false:N \g_tmpa_bool }
6725       {
6726         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6727         { \bool_gset_false:N \g_tmpa_bool }
6728       }
6729     }
6730   }
6731 }

```

24 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```

6732 \cs_new_protected:Npn \@@_compute_corners:
6733 {

```

The list `\l_@@_corners_cells_clist` will be the list of all the empty cells (and not in a block) considered in the corners of the array. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

6734   \clist_clear:N \l_@@_corners_cells_clist
6735   \clist_map_inline:Nn \l_@@_corners_clist
6736   {
6737     \str_case:nnF { ##1 }
6738     {
6739       { NW }
6740       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6741       { NE }
6742       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6743       { SW }
6744       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6745       { SE }
6746       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }

```

```

6747     }
6748     { \@@_error:nn { bad-corner } { ##1 } }
6749 }

```

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

```

6750 \clist_if_empty:NF \l_@@_corners_cells_clist
6751 {

```

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 colors the `rows`, `columns` and `cells` must not color the cells in the corners.

```

6752 \tl_gput_right:Ne \g_@@_aux_tl
6753 {
6754     \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6755     { \l_@@_corners_cells_clist }
6756 }
6757 }
6758 }

```

“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_clist`.

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.

```

6759 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6760 {

```

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.

```

6761 \bool_set_false:N \l_tmpa_bool
6762 \int_zero_new:N \l_@@_last_empty_row_int
6763 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6764 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6765 {
6766     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6767     \bool_lazy_or:nnTF
6768     {
6769         \cs_if_exist_p:c
6770         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6771     }
6772     \l_tmpb_bool
6773     { \bool_set_true:N \l_tmpa_bool }
6774     {
6775         \bool_if:NF \l_tmpa_bool
6776         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6777     }
6778 }

```

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

```

6779 \bool_set_false:N \l_tmpa_bool
6780 \int_zero_new:N \l_@@_last_empty_column_int
6781 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6782 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6783 {

```

```

6784 \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6785 \bool_lazy_or:nnTF
6786 \l_tmpb_bool
6787 {
6788   \cs_if_exist_p:c
6789   { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6790 }
6791 { \bool_set_true:N \l_tmpa_bool }
6792 {
6793   \bool_if:NF \l_tmpa_bool
6794   { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6795 }
6796 }

```

Now, we loop over the rows.

```

6797 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6798 {

```

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

```

6799   \bool_set_false:N \l_tmpa_bool
6800   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6801   {
6802     \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6803     \bool_lazy_or:nnTF
6804     \l_tmpb_bool
6805     {
6806       \cs_if_exist_p:c
6807       { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6808     }
6809     { \bool_set_true:N \l_tmpa_bool }
6810     {
6811       \bool_if:NF \l_tmpa_bool
6812       {
6813         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6814         \clist_put_right:Nn
6815         \l_@@_corners_cells_clist
6816         { ##1 - #####1 }
6817       }
6818     }
6819   }
6820 }
6821 }

```

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`).

```

6822 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6823 {
6824   \int_set:Nn \l_tmpa_int { #1 }
6825   \int_set:Nn \l_tmpb_int { #2 }
6826   \bool_set_false:N \l_tmpb_bool
6827   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6828   { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6829 }
6830 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6 #7
6831 {
6832   \int_compare:nNf { #3 } > { #1 }
6833   {
6834     \int_compare:nNf { #1 } > { #5 }
6835     {
6836       \int_compare:nNf { #4 } > { #2 }
6837       {
6838         \int_compare:nNf { #2 } > { #6 }

```



```

6839         { \bool_set_true:N \l_tmpb_bool }
6840     }
6841 }
6842 }
6843 }

```

25 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.

```

6844 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6845 \keys_define:nn { nicematrix / NiceMatrixBlock }
6846 {
6847     auto-columns-width .code:n =
6848     {
6849         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6850         \dim_gzero_new:N \g_@@_max_cell_width_dim
6851         \bool_set_true:N \l_@@_auto_columns_width_bool
6852     }
6853 }

6854 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6855 {
6856     \int_gincr:N \g_@@_NiceMatrixBlock_int
6857     \dim_zero:N \l_@@_columns_width_dim
6858     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6859     \bool_if:NT \l_@@_block_auto_columns_width_bool
6860     {
6861         \cs_if_exist:cT
6862         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6863         {
6864             \dim_set:Nn \l_@@_columns_width_dim
6865             {
6866                 \use:c
6867                 { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6868             }
6869         }
6870     }
6871 }

```

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).

```

6872 {
6873     \legacy_if:nTF { measuring@ }

```

If {NiceMatrixBlock} is used in an environment of amsmath such as {align}: cf. question 694957 on TeX StackExchange. The most important line in that case is the following one.

```

6874     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6875     {
6876         \bool_if:NT \l_@@_block_auto_columns_width_bool
6877         {
6878             \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6879             \iow_shipout:Ne \@mainaux
6880             {
6881                 \cs_gset:cpn
6882                 { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

6883         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6884     }
6885     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6886 }
6887 }
6888 \ignorespacesafterend
6889 }

```

26 The extra nodes

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).

```

6890 \cs_new_protected:Npn \@@_create_extra_nodes:
6891 {
6892     \bool_if:nTF \l_@@_medium_nodes_bool
6893     {
6894         \bool_if:NTF \l_@@_large_nodes_bool
6895         \@@_create_medium_and_large_nodes:
6896         \@@_create_medium_nodes:
6897     }
6898     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6899 }

```

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`.

```

6900 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6901 {
6902     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6903     {
6904         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6905         \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6906         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6907         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6908     }
6909     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6910     {
6911         \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6912         \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6913         \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6914         \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6915     }

```

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

```

6916   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6917   {
6918     \int_step_variable:nnNn
6919     \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.

```

6920     {
6921       \cs_if_exist:cT
6922       { 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`.

```

6923     {
6924       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6925       \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6926       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6927       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6928       {
6929         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6930         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6931       }

```

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`.

```

6932       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6933       \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6934       { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6935       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6936       {
6937         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6938         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6939       }
6940     }
6941   }
6942 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6943   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6944   {
6945     \dim_compare:nNnT
6946     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6947     {
6948       \@@_qpoint:n { row - \@@_i: - base }
6949       \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6950       \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6951     }
6952   }
6953   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6954   {
6955     \dim_compare:nNnT
6956     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6957     {
6958       \@@_qpoint:n { col - \@@_j: }
6959       \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6960       \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6961     }
6962   }
6963 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6964 \cs_new_protected:Npn \@@_create_medium_nodes:
6965 {
6966   \pgfpicture
6967     \pgfrememberpicturepositiononpagetrue
6968     \pgf@relevantforpicturesizefalse
6969     \@@_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”.

```

6970     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6971     \@@_create_nodes:
6972   \endpgfpicture
6973 }

```

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:`.

```

6974 \cs_new_protected:Npn \@@_create_large_nodes:
6975 {
6976   \pgfpicture
6977     \pgfrememberpicturepositiononpagetrue
6978     \pgf@relevantforpicturesizefalse
6979     \@@_computations_for_medium_nodes:
6980     \@@_computations_for_large_nodes:
6981     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6982     \@@_create_nodes:
6983   \endpgfpicture
6984 }
6985 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6986 {
6987   \pgfpicture
6988     \pgfrememberpicturepositiononpagetrue
6989     \pgf@relevantforpicturesizefalse
6990     \@@_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”.

```

6991     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6992     \@@_create_nodes:
6993     \@@_computations_for_large_nodes:
6994     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6995     \@@_create_nodes:
6996   \endpgfpicture
6997 }

```

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.

```

6998 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6999 {
7000   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7001   \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

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`.

```

7002   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7003   {
7004     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7005     {
7006     (
7007         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7008         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7009     )
7010     / 2
7011     }
7012     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7013     { l_@@_row_\@@_i: _ min_dim }
7014 }
7015 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7016 {
7017     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7018     {
7019     (
7020         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7021         \dim_use:c
7022         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7023     )
7024     / 2
7025     }
7026     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7027     { l_@@_column _ \@@_j: _ max _ dim }
7028 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7029     \dim_sub:cn
7030     { l_@@_column _ 1 _ min _ dim }
7031     \l_@@_left_margin_dim
7032     \dim_add:cn
7033     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7034     \l_@@_right_margin_dim
7035 }

```

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).

```

7036 \cs_new_protected:Npn \@@_create_nodes:
7037 {
7038     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7039     {
7040         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7041         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7042         \@@_pgf_rect_node:nnnnn
7043         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7044         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7045         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7046         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7047         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7048         \str_if_empty:NF \l_@@_name_str
7049         {
7050             \pgfnodealias
7051             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7052             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7053         }
7054     }
7055 }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with $n > 1$ was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of n .

```

7056     \seq_map_pairwise_function:NNN
7057     \g_@@_multicolumn_cells_seq
7058     \g_@@_multicolumn_sizes_seq
7059     \@@_node_for_multicolumn:nn
7060 }

7061 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7062 {
7063   \cs_set_nopar:Npn \@@_i: { #1 }
7064   \cs_set_nopar:Npn \@@_j: { #2 }
7065 }
```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{n}{...}{...}` was issued in the format i - j and the second is the value of n (the length of the “multi-cell”).

```

7066 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7067 {
7068   \@@_extract_coords_values: #1 \q_stop
7069   \@@_pgf_rect_node:nnnnn
7070   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7071   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7072   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7073   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7074   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7075   \str_if_empty:NF \l_@@_name_str
7076   {
7077     \pgfnodealias
7078     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7079     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7080   }
7081 }
```

27 The blocks

The following code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass (in the cell of the array).

```

7082 \keys_define:nn { nicematrix / Block / FirstPass }
7083 {
7084   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7085           \bool_set_true:N \l_@@_p_block_bool ,
7086   j .value_forbidden:n = true ,
7087   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7088   l .value_forbidden:n = true ,
7089   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7090   r .value_forbidden:n = true ,
7091   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7092   c .value_forbidden:n = true ,
7093   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7094   L .value_forbidden:n = true ,
7095   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
```

```

7096 R .value_forbidden:n = true ,
7097 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7098 C .value_forbidden:n = true ,
7099 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7100 t .value_forbidden:n = true ,
7101 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7102 T .value_forbidden:n = true ,
7103 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7104 b .value_forbidden:n = true ,
7105 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7106 B .value_forbidden:n = true ,
7107 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7108 m .value_forbidden:n = true ,
7109 v-center .meta:n = m ,
7110 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7111 p .value_forbidden:n = true ,
7112 color .code:n =
7113   \@@_color:n { #1 }
7114   \tl_set_rescan:Nnn
7115     \l_@@_draw_tl
7116     { \char_set_catcode_other:N ! }
7117     { #1 } ,
7118 color .value_required:n = true ,
7119 respect-arraystretch .code:n =
7120   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7121 respect-arraystretch .value_forbidden:n = true ,
7122 }

```

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.

```

7123 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7124 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7125 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not been provided by the user, you use 1-1 (that is to say a block of only one cell).

```

7126   \peek_remove_spaces:n
7127   {
7128     \tl_if_blank:nTF { #2 }
7129     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7130     {
7131       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7132       \@@_Block_i_czech \@@_Block_i
7133       #2 \q_stop
7134     }
7135     { #1 } { #3 } { #4 }
7136   }
7137 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7138 \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`.

```

7139 {
7140   \char_set_catcode_active:N -
7141   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7142 }

```

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 before the composition of the block and #5 is the label (=content) of the block.

```
7143 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7144 {
```

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).

```
7145 \bool_lazy_or:nnTF
7146 { \tl_if_blank_p:n { #1 } }
7147 { \str_if_eq_p:ee { * } { #1 } }
7148 { \int_set:Nn \l_tmpa_int { 100 } }
7149 { \int_set:Nn \l_tmpa_int { #1 } }
7150 \bool_lazy_or:nnTF
7151 { \tl_if_blank_p:n { #2 } }
7152 { \str_if_eq_p:ee { * } { #2 } }
7153 { \int_set:Nn \l_tmpb_int { 100 } }
7154 { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7155 \int_compare:nNnTF \l_tmpb_int = \c_one_int
7156 {
7157   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7158   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7159   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7160 }
7161 { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```
7162 \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7163 \tl_set:Ne \l_tmpa_tl
7164 {
7165   { \int_use:N \c@iRow }
7166   { \int_use:N \c@jCol }
7167   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7168   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7169 }
```

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}`.

We have different treatments when the key `p` is used and when the block is mono-column or mono-row, etc. That's why we have several macros: `\@@_Block_iv:nnnnn`, `\@@_Block_v:nnnnn`, `\@@_Block_vi:nnnn`, etc. (the five arguments of those macros are provided by curryfication).

```
7170 \bool_set_false:N \l_tmpa_bool
7171 \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```
7172 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7173 \bool_case:nF
7174 {
7175   \l_tmpa_bool { \@@_Block_vii:eennn }
7176   \l_@@_p_block_bool { \@@_Block_vi:eennn }
```


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.

```

7177         \l_@@_X_bool                                { \@@_Block_v:eennn }
7178         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7179         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7180         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7181     }
7182     { \@@_Block_v:eennn }
7183     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7184 }

```

The following macro is for the case of a \Block which is mono-row or mono-column (or both) and don't use the key p. 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) with \@@_draw_blocks: and above all \@@_Block_v:nnnnnn which will do the main job.

#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 potential math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7185 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7186 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7187 {
7188     \int_gincr:N \g_@@_block_box_int
7189     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7190     {
7191         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7192         {
7193             \@@_actually_diagbox:nnnnnn
7194             { \int_use:N \c@iRow }
7195             { \int_use:N \c@jCol }
7196             { \int_eval:n { \c@iRow + #1 - 1 } }
7197             { \int_eval:n { \c@jCol + #2 - 1 } }
7198             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7199             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7200         }
7201     }
7202     \box_gclear_new:c
7203     { \g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the \Block in a TeX box. *Be careful:* if after the construction of the box, the boolean \g_@@_rotate_bool is raised (which means that the command \rotate was present in the content of the \Block) we will rotate the box but also, maybe, change the position of the baseline!

```

7204     \hbox_gset:cn
7205     { \g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7206     {

```

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}).

```

7207         \tl_if_empty:NTF \l_@@_color_tl
7208         { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7209         { \@@_color:o \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.

```

7210 \int_compare:nNnT { #1 } = \c_one_int
7211 {
7212     \int_if_zero:nTF \c@iRow
7213     {

```

In the following code, the value of `code-for-first-row` contains a `\Block` (in order to have the “first row” centered). But, that block will be executed, since it is entirely contained in the first row, the value of `code-for-first-row` will be inserted once again... with the same command `\Block`. That’s why we have to nullify the command `\Block`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\\
-2 & 3 & -4 & 5 & \\\
3 & -4 & 5 & -6 & \\\
-4 & 5 & -6 & 7 & \\\
5 & -6 & 7 & -8 & \\\
\end{bNiceMatrix}$

```

```

7214 \cs_set_eq:NN \Block \@@_NullBlock:
7215 \l_@@_code_for_first_row_tl
7216 }
7217 {
7218     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7219     {
7220         \cs_set_eq:NN \Block \@@_NullBlock:
7221         \l_@@_code_for_last_row_tl
7222     }
7223 }
7224 \g_@@_row_style_tl
7225 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7226 \@@_reset_arraystretch:
7227 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7228 #4

```

We adjust `\l_@@_hpos_block_str` when `\rotate` has been used (in the cell where the command `\Block` is used but maybe in **#4**, `\RowStyle`, `code-for-first-row`, etc.).

```

7229 \@@_adjust_hpos_rotate:

```

The boolean `\g_@@_rotate_bool` will be also considered *after the composition of the box* (in order to rotate the box).

Remind that we are in the command of composition of the box of the block. Previously, we have only done some tuning. Now, we will actually compose the content with a `{tabular}`, an `{array}` or a `{minipage}`.

```

7230 \bool_if:NTF \l_@@_tabular_bool
7231 {
7232     \bool_lazy_all:nTF
7233     {
7234         { \int_compare_p:nNn { #2 } = \c_one_int }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```

7235         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7236         { ! \g_@@_rotate_bool }
7237     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7238     {
7239         \use:e
7240     {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7241         \exp_not:N \begin { minipage }%
7242         [ \str_lowercase:o \l_@@_vpos_block_str ]
7243         { \l_@@_col_width_dim }
7244         \str_case:on \l_@@_hpos_block_str
7245         { c \centering r \raggedleft l \raggedright }
7246     }
7247     #5
7248     \end { minipage }
7249 }

```

In the other cases, we use a `{tabular}`.

```

7250     {
7251         \use:e
7252     {
7253         \exp_not:N \begin { tabular }%
7254         [ \str_lowercase:o \l_@@_vpos_block_str ]
7255         { @ { } \l_@@_hpos_block_str @ { } }
7256     }
7257     #5
7258     \end { tabular }
7259 }
7260 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7261     {
7262         \c_math_toggle_token
7263         \use:e
7264     {
7265         \exp_not:N \begin { array }%
7266         [ \str_lowercase:o \l_@@_vpos_block_str ]
7267         { @ { } \l_@@_hpos_block_str @ { } }
7268     }
7269     #5
7270     \end { array }
7271     \c_math_toggle_token
7272 }
7273 }

```

The box which will contain the content of the block has now been composed.

If there were `\rotate` (which raises `\g_@@_rotate_bool`) in the content of the `\Block`, we do a rotation of the box (and we also adjust the baseline of the rotated box).

```

7274     \bool_if:NT \g_@@_rotate_bool \@@_rotate_box_of_block:

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

7275     \int_compare:nNnT { #2 } = \c_one_int
7276     {
7277         \dim_gset:Nn \g_@@_blocks_wd_dim
7278         {
7279             \dim_max:nn
7280             \g_@@_blocks_wd_dim

```

```

7281         {
7282             \box_wd:c
7283             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7284         }
7285     }
7286 }

```

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, excepted when the block uses explicitly an option of vertical position.

```

7287 \bool_lazy_and:nnT
7288 { \int_compare_p:nNn { #1 } = \c_one_int }

```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7289 { \str_if_empty_p:N \l_@@_vpos_block_str }
7290 {
7291     \dim_gset:Nn \g_@@_blocks_ht_dim
7292     {
7293         \dim_max:nn
7294         \g_@@_blocks_ht_dim
7295         {
7296             \box_ht:c
7297             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7298         }
7299     }
7300     \dim_gset:Nn \g_@@_blocks_dp_dim
7301     {
7302         \dim_max:nn
7303         \g_@@_blocks_dp_dim
7304         {
7305             \box_dp:c
7306             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7307         }
7308     }
7309 }
7310 \seq_gput_right:Ne \g_@@_blocks_seq
7311 {
7312     \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.

```

7313 {
7314     \exp_not:n { #3 } ,
7315     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7316 \bool_if:NT \g_@@_rotate_bool
7317 {
7318     \bool_if:NTF \g_@@_rotate_c_bool
7319     { m }
7320     { \int_compare:nNnT \c_iRow = \l_@@_last_row_int T }
7321 }
7322 }
7323 {
7324     \box_use_drop:c
7325     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7326 }
7327 }
7328 \bool_set_false:N \g_@@_rotate_c_bool
7329 }

```

```

7330 \cs_new:Npn \@@_adjust_hpos_rotate:
7331 {
7332   \bool_if:NT \g_@@_rotate_bool
7333   {
7334     \str_set:Ne \l_@@_hpos_block_str
7335     {
7336       \bool_if:NTF \g_@@_rotate_c_bool
7337       { c }
7338       {
7339         \str_case:onF \l_@@_vpos_block_str
7340         { b l B l t r T r }
7341         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7342       }
7343     }
7344   }
7345 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7346 \cs_new_protected:Npn \@@_rotate_box_of_block:
7347 {
7348   \box_grotate:cn
7349   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7350   { 90 }
7351   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7352   {
7353     \vbox_gset_top:cn
7354     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7355     {
7356       \skip_vertical:n { 0.8 ex }
7357       \box_use:c
7358       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7359     }
7360   }
7361   \bool_if:NT \g_@@_rotate_c_bool
7362   {
7363     \hbox_gset:cn
7364     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7365     {
7366       \c_math_toggle_token
7367       \vcenter
7368       {
7369         \box_use:c
7370         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7371       }
7372       \c_math_toggle_token
7373     }
7374   }
7375 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). 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 (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#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 before the composition of the block and #5 is the label (=content) of the block.

```

7376 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7377 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7378 {
7379   \seq_gput_right:Ne \g_@@_blocks_seq
7380   {

```

```

7381 \l_tmpa_tl
7382 { \exp_not:n { #3 } }
7383 {
7384   \bool_if:NTF \l_@@_tabular_bool
7385   {
7386     \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7387   \@@_reset_arraystretch:
7388   \exp_not:n
7389   {
7390     \dim_zero:N \extrarowheight
7391     #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).

```

7392   \bool_if:NT \c_@@_testphase_table_bool
7393   { \tag_stop:n { table } }
7394   \use:e
7395   {
7396     \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7397     { @ { } \l_@@_hpos_block_str @ { } }
7398   }
7399   #5
7400   \end { tabular }
7401 }
7402 \group_end:
7403 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7404 {
7405   \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7406   \@@_reset_arraystretch:
7407   \exp_not:n
7408   {
7409     \dim_zero:N \extrarowheight
7410     #4
7411     \c_math_toggle_token
7412     \use:e
7413     {
7414       \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7415       { @ { } \l_@@_hpos_block_str @ { } }
7416     }
7417     #5
7418     \end { array }
7419     \c_math_toggle_token
7420   }
7421   \group_end:
7422 }
7423 }
7424 }
7425 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7426 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7427 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7428 {
7429   \seq_gput_right:Ne \g_@@_blocks_seq
7430   {

```

```

7431     \l_tmpa_tl
7432     { \exp_not:n { #3 } }
7433     {
7434         \group_begin:
7435         \exp_not:n { #4 #5 }
7436         \group_end:
7437     }
7438 }
7439 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7440 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7441 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7442 {
7443     \seq_gput_right:Ne \g_@@_blocks_seq
7444     {
7445         \l_tmpa_tl
7446         { \exp_not:n { #3 } }
7447         { \exp_not:n { #4 #5 } }
7448     }
7449 }

```

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).

```

7450 \keys_define:nn { nicematrix / Block / SecondPass }
7451 {
7452     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7453     ampersand-in-blocks .default:n = true ,
7454     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7455     tikz .code:n =
7456         \IfPackageLoadedTF { tikz }
7457         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7458         { \@@_error:n { tikz~key~without~tikz } } ,
7459     tikz .value_required:n = true ,
7460     fill .code:n =
7461         \tl_set_rescan:Nnn
7462         \l_@@_fill_tl
7463         { \char_set_catcode_other:N ! }
7464         { #1 } ,
7465     fill .value_required:n = true ,
7466     opacity .tl_set:N = \l_@@_opacity_tl ,
7467     opacity .value_required:n = true ,
7468     draw .code:n =
7469         \tl_set_rescan:Nnn
7470         \l_@@_draw_tl
7471         { \char_set_catcode_other:N ! }
7472         { #1 } ,
7473     draw .default:n = default ,
7474     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7475     rounded-corners .default:n = 4 pt ,
7476     color .code:n =
7477         \@@_color:n { #1 }
7478         \tl_set_rescan:Nnn
7479         \l_@@_draw_tl
7480         { \char_set_catcode_other:N ! }
7481         { #1 } ,
7482     borders .clist_set:N = \l_@@_borders_clist ,
7483     borders .value_required:n = true ,
7484     hvlines .meta:n = { vlines , hlines } ,

```

```

7485   vlines .bool_set:N = \l_@@_vlines_block_bool,
7486   vlines .default:n = true ,
7487   hlines .bool_set:N = \l_@@_hlines_block_bool,
7488   hlines .default:n = true ,
7489   line-width .dim_set:N = \l_@@_line_width_dim ,
7490   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7491   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7492             \bool_set_true:N \l_@@_p_block_bool ,
7493   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7494   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7495   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7496   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7497             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7498   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7499             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7500   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7501             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7502   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7503   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7504   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7505   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7506   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7507   m .value_forbidden:n = true ,
7508   v-center .meta:n = m ,
7509   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7510   p .value_forbidden:n = true ,
7511   name .tl_set:N = \l_@@_block_name_str ,
7512   name .value_required:n = true ,
7513   name .initial:n = ,
7514   respect-arraystretch .code:n =
7515     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7516   respect-arraystretch .value_forbidden:n = true ,
7517   transparent .bool_set:N = \l_@@_transparent_bool ,
7518   transparent .default:n = true ,
7519   transparent .initial:n = false ,
7520   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7521 }

```

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.

```

7522 \cs_new_protected:Npn \@@_draw_blocks:
7523 {
7524   \bool_if:NTF \c_@@_tagging_array_bool
7525   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7526   { \cs_set_eq:NN \ialign \@@_old_ialign: }
7527   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7528 }
7529 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7530 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7531 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7532   \int_zero_new:N \l_@@_last_row_int
7533   \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.

```

7534 \int_compare:nNnTF { #3 } > { 99 }
7535   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7536   { \int_set:Nn \l_@@_last_row_int { #3 } }
7537 \int_compare:nNnTF { #4 } > { 99 }
7538   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7539   { \int_set:Nn \l_@@_last_col_int { #4 } }
7540 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7541   {
7542     \bool_lazy_and:nnTF
7543       \l_@@_preamble_bool
7544       {
7545         \int_compare_p:n
7546           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7547       }
7548       {
7549         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7550         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7551         \@@_msg_redirect_name:nn { columns-not-used } { none }
7552       }
7553       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7554   }
7555   {
7556     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7557     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7558     {
7559       \@@_Block_v:nneenn
7560       { #1 }
7561       { #2 }
7562       { \int_use:N \l_@@_last_row_int }
7563       { \int_use:N \l_@@_last_col_int }
7564       { #5 }
7565       { #6 }
7566     }
7567   }
7568 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. #1 is the first row of the block; #2 is the first column of the block; #3 is the last row of the block; #4 is the last column of the block; #5 is a list of `key=value` options; #6 is the label

```

7569 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7570 {

```

The group is for the keys.

```

7571 \group_begin:
7572 \int_compare:nNnT { #1 } = { #3 }
7573   { \str_set:Nn \l_@@_vpos_block_str { t } }
7574 \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```

7575 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7576 \bool_lazy_and:nnT
7577   \l_@@_vlines_block_bool
7578   { ! \l_@@_ampersand_bool }
7579   {
7580     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7581     {
7582       \@@_vlines_block:nnn
7583       { \exp_not:n { #5 } }
7584       { #1 - #2 }
7585       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }

```

```

7586     }
7587   }
7588   \bool_if:NT \l_@@_hlines_block_bool
7589   {
7590     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7591     {
7592       \@@_hlines_block:nnn
7593       { \exp_not:n { #5 } }
7594       { #1 - #2 }
7595       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7596     }
7597   }
7598   \bool_if:NF \l_@@_transparent_bool
7599   {
7600     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7601     {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7602       \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7603       { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7604     }
7605   }

```

```

7606   \tl_if_empty:NF \l_@@_draw_tl
7607   {
7608     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7609     { \@@_error:n { hlines~with~color } }
7610     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7611     {
7612       \@@_stroke_block:nnn

```

#5 are the options

```

7613       { \exp_not:n { #5 } }
7614       { #1 - #2 }
7615       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7616     }
7617     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7618     { { #1 } { #2 } { #3 } { #4 } }
7619   }
7620   \clist_if_empty:NF \l_@@_borders_clist
7621   {
7622     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7623     {
7624       \@@_stroke_borders_block:nnn
7625       { \exp_not:n { #5 } }
7626       { #1 - #2 }
7627       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7628     }
7629   }
7630   \tl_if_empty:NF \l_@@_fill_tl
7631   {
7632     \tl_if_empty:NF \l_@@_opacity_tl
7633     {
7634       \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7635         {
7636           \tl_set:Ne \l_@@_fill_tl
7637           {
7638             [ opacity = \l_@@_opacity_tl ,
7639             \tl_tail:o \l_@@_fill_tl
7640           }
7641         }

```

```

7642         {
7643             \tl_set:Nc \l_@@_fill_tl
7644             { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7645         }
7646     }
7647     \tl_gput_right:Nc \g_@@_pre_code_before_tl
7648     {
7649         \exp_not:N \roundedrectanglecolor
7650         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7651             { \l_@@_fill_tl }
7652             { { \l_@@_fill_tl } }
7653             { #1 - #2 }
7654             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7655             { \dim_use:N \l_@@_rounded_corners_dim }
7656         ]
7657     }
7658     \seq_if_empty:NF \l_@@_tikz_seq
7659     {
7660         \tl_gput_right:Nc \g_nicematrix_code_before_tl
7661         {
7662             \@@_block_tikz:nnnnn
7663             { \seq_use:Nn \l_@@_tikz_seq { , } }
7664             { #1 }
7665             { #2 }
7666             { \int_use:N \l_@@_last_row_int }
7667             { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of list of Tikz keys.

```

7668         }
7669     }

7670     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7671     {
7672         \tl_gput_right:Nc \g_@@_pre_code_after_tl
7673         {
7674             \@@_actually_diagbox:nnnnnn
7675             { #1 }
7676             { #2 }
7677             { \int_use:N \l_@@_last_row_int }
7678             { \int_use:N \l_@@_last_col_int }
7679             { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7680         }
7681     }

```

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
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7682 \pgfpicture
7683 \pgfrememberpicturepositiononpagetrue
7684 \pgf@relevantforpicturesizefalse
7685 \@@_qpoint:n { row - #1 }
7686 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7687 \@@_qpoint:n { col - #2 }
7688 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7689 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7690 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7691 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7692 \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.

```

7693 \@@_pgf_rect_node:nnnnn
7694 { \@@_env: - #1 - #2 - block }
7695 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7696 \str_if_empty:NF \l_@@_block_name_str
7697 {
7698   \pgfnodealias
7699   { \@@_env: - \l_@@_block_name_str }
7700   { \@@_env: - #1 - #2 - block }
7701   \str_if_empty:NF \l_@@_name_str
7702   {
7703     \pgfnodealias
7704     { \l_@@_name_str - \l_@@_block_name_str }
7705     { \@@_env: - #1 - #2 - block }
7706   }
7707 }

```

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.

```

7708 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7709 {
7710   \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.

```

7711 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7712 {

```

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.

```

7713 \cs_if_exist:cT
7714 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7715 {
7716   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7717   {
7718     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7719     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7720   }
7721 }
7722 }

```

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.

```

7723 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7724 {
7725   \@@_qpoint:n { col - #2 }
7726   \dim_set_eq:NN \l_tmpb_dim \pgf@x

```

```

7727     }
7728     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7729     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7730     {
7731         \cs_if_exist:cT
7732         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7733         {
7734             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7735             {
7736                 \pgfpointanchor
7737                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7738                 { east }
7739                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7740             }
7741         }
7742     }
7743     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7744     {
7745         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7746         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7747     }
7748     \@@_pgf_rect_node:nnnnn
7749     { \@@_env: - #1 - #2 - block - short }
7750     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7751 }

```

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.

```

7752     \bool_if:NT \l_@@_medium_nodes_bool
7753     {
7754         \@@_pgf_rect_node:nnn
7755         { \@@_env: - #1 - #2 - block - medium }
7756         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7757         {
7758             \pgfpointanchor
7759             { \@@_env:
7760               - \int_use:N \l_@@_last_row_int
7761               - \int_use:N \l_@@_last_col_int - medium
7762             }
7763             { south-east }
7764         }
7765     }
7766     \endpgfpicture

```

```

7767     \bool_if:NTF \l_@@_ampersand_bool
7768     {
7769         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7770         \int_zero_new:N \l_@@_split_int
7771         \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7772         \pgfpicture
7773         \pgfrememberpicturepositiononpagetrue
7774         \pgf@relevantforpicturesizefalse
7775         \@@_qpoint:n { row - #1 }
7776         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7777         \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7778         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7779         \@@_qpoint:n { col - #2 }
7780         \dim_set_eq:NN \l_tmpa_dim \pgf@x
7781         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7782         \dim_set:Nn \l_tmpb_dim
7783         { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7784         \bool_lazy_or:nnT
7785         \l_@@_vlines_block_bool

```

```

7786 { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7787 {
7788   \int_step_inline:nn { \l_@@_split_int - 1 }
7789   {
7790     \pgfpathmoveto
7791     {
7792       \pgfpoint
7793       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7794       \l_@@_tmpc_dim
7795     }
7796     \pgfpathlineto
7797     {
7798       \pgfpoint
7799       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7800       \l_@@_tmpd_dim
7801     }
7802     \CT@arc@
7803     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7804     \pgfsetrectcap
7805     \pgfusepathqstroke
7806   }
7807 }
7808 \@@_qpoint:n { row - #1 - base }
7809 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7810 \int_step_inline:nn \l_@@_split_int
7811 {
7812   \group_begin:
7813   \dim_set:Nn \col@sep
7814   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7815   \pgftransformshift
7816   {
7817     \pgfpoint
7818     {
7819       \str_case:on \l_@@_hpos_block_str
7820       {
7821         l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep }
7822         c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
7823         r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
7824       }
7825     }
7826     { \l_@@_tmpc_dim }
7827   }
7828   \pgfset
7829   {
7830     inner~xsep = \c_zero_dim ,
7831     inner~ysep = \c_zero_dim
7832   }
7833   \pgfnode
7834   { rectangle }
7835   {
7836     \str_case:on \l_@@_hpos_block_str
7837     {
7838       c { base }
7839       l { base~west }
7840       r { base~east }
7841     }
7842   }
7843   { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
7844   \group_end:
7845 }
7846 \endpgfpicture
7847 }

```

Now the case where there is no ampersand & in the content of the block.

```

7848 {
7849   \bool_if:NTF \l_@@_p_block_bool
7850   {

```

When the final user has used the key p, we have to compute the width.

```

7851     \pgfpicture
7852     \pgfrememberpicturepositiononpagetrue
7853     \pgf@relevantforpicturesizefalse
7854     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7855     {
7856       \@@_qpoint:n { col - #2 }
7857       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7858       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7859     }
7860     {
7861       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7862       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7863       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7864     }
7865     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7866     \endpgfpicture
7867     \hbox_set:Nn \l_@@_cell_box
7868     {
7869       \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7870       { \g_tmpb_dim }
7871       \str_case:on \l_@@_hpos_block_str
7872       { c \centering r \raggedleft l \raggedright j { } }
7873       #6
7874       \end { minipage }
7875     }
7876   }
7877   { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7878   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Now, we will put the label of the block. We recall that \l_@@_vpos_block_str is empty when the user has not used a key for the vertical position of the block.

```

7879     \pgfpicture
7880     \pgfrememberpicturepositiononpagetrue
7881     \pgf@relevantforpicturesizefalse
7882     \bool_lazy_any:nTF
7883     {
7884       { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7885       { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7886       { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7887       { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7888     }
7889     {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7890       \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the last column, we must put the block as if it was with the key l.

```

7891       \bool_if:nT \g_@@_last_col_found_bool
7892       {
7893         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7894         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7895       }

```

\l_tmpa_tl will contain the anchor of the PGF node which will be used.

```

7896       \tl_set:Ne \l_tmpa_tl
7897       {
7898         \str_case:on \l_@@_vpos_block_str
7899         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7900         { } { % added 2024-06-29
7901             \str_case:on \l_@@_hpos_block_str
7902             {
7903                 c { center }
7904                 l { west }
7905                 r { east }
7906                 j { center }
7907             }
7908         }
7909     c {
7910         \str_case:on \l_@@_hpos_block_str
7911         {
7912             c { center }
7913             l { west }
7914             r { east }
7915             j { center }
7916         }
7917     }
7918     T {
7919         \str_case:on \l_@@_hpos_block_str
7920         {
7921             c { north }
7922             l { north-west }
7923             r { north-east }
7924             j { north }
7925         }
7926     }
7927     B {
7928         \str_case:on \l_@@_hpos_block_str
7929         {
7930             c { south }
7931             l { south-west }
7932             r { south-east }
7933             j { south }
7934         }
7935     }
7936     }
7937     }
7938     }
7939     }
7940     }
7941     \pgftransformshift
7942     {
7943         \pgfpointanchor
7944         {
7945             \@@_env: - #1 - #2 - block
7946             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7947         }
7948         { \l_tmpa_tl }
7949     }
7950     \pgfset
7951     {
7952         inner~xsep = \c_zero_dim ,
7953         inner~ysep = \c_zero_dim
7954     }
7955     \pgfnode
7956     { rectangle }
7957     { \l_tmpa_tl }
7958     { \box_use_drop:N \l_@@_cell_box } { } { } { }
7959 }
```


End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

7960     {
7961         \pgfextracty \l_tmpa_dim
7962         {
7963             \l_@@_qpoint:n
7964             {
7965                 row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7966                 - base
7967             }
7968         }
7969         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

7970     \pgfpointanchor
7971     {
7972         \l_@@_env: - #1 - #2 - block
7973         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7974     }
7975     {
7976         \str_case:on \l_@@_hpos_block_str
7977         {
7978             c { center }
7979             l { west }
7980             r { east }
7981             j { center }
7982         }
7983     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7984     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7985     \pgfset { inner~sep = \c_zero_dim }
7986     \pgfnode
7987     { rectangle }
7988     {
7989         \str_case:on \l_@@_hpos_block_str
7990         {
7991             c { base }
7992             l { base~west }
7993             r { base~east }
7994             j { base }
7995         }
7996     }
7997     { \box_use_drop:N \l_@@_cell_box } { } { }
7998 }
7999 \endpgfpicture
8000 }
8001 \group_end:
8002 }

```

The first argument of `\l_@@_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).

```

8003 \cs_new_protected:Npn \l_@@_stroke_block:nnn #1 #2 #3
8004 {
8005     \group_begin:
8006     \tl_clear:N \l_@@_draw_tl
8007     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8008     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8009     \pgfpicture
8010     \pgfrememberpicturepositiononpagetrue
8011     \pgf@relevantforpicturesizefalse
8012     \tl_if_empty:NF \l_@@_draw_tl
8013     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8014     \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
8015     { \CT@arc@ }
8016     { \@@_color:o \l_@@_draw_tl }
8017   }
8018   \pgfsetcornersarced
8019   {
8020     \pgfpoint
8021     { \l_@@_rounded_corners_dim }
8022     { \l_@@_rounded_corners_dim }
8023   }
8024   \@@_cut_on_hyphen:w #2 \q_stop
8025   \int_compare:nNnF \l_tmpa_tl > \c@iRow
8026   {
8027     \int_compare:nNnF \l_tmpb_tl > \c@jCol
8028     {
8029       \@@_qpoint:n { row - \l_tmpa_tl }
8030       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8031       \@@_qpoint:n { col - \l_tmpb_tl }
8032       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8033       \@@_cut_on_hyphen:w #3 \q_stop
8034       \int_compare:nNnT \l_tmpa_tl > \c@iRow
8035       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8036       \int_compare:nNnT \l_tmpb_tl > \c@jCol
8037       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8038       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8039       \dim_set_eq:NN \l_tmpa_dim \pgf@y
8040       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8041       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8042       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8043       \pgfpathrectanglecorners
8044       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8045       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8046       \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8047       { \pgfusepathqstroke }
8048       { \pgfusepath { stroke } }
8049     }
8050   }
8051   \endpgfpicture
8052   \group_end:
8053 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8054 \keys_define:nn { nicematrix / BlockStroke }
8055 {
8056   color .tl_set:N = \l_@@_draw_tl ,
8057   draw .code:n =
8058     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8059   draw .default:n = default ,
8060   line-width .dim_set:N = \l_@@_line_width_dim ,
8061   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8062   rounded-corners .default:n = 4 pt
8063 }

```

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).

```

8064 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8065 {
8066   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8067   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8068   \@@_cut_on_hyphen:w #2 \q_stop

```

```

8069 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8070 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8071 \@@_cut_on_hyphen:w #3 \q_stop
8072 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8073 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8074 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8075 {
8076   \use:e
8077   {
8078     \@@_vline:n
8079     {
8080       position = ##1 ,
8081       start = \l_@@_tmpc_tl ,
8082       end = \int_eval:n { \l_tmpa_tl - 1 } ,
8083       total-width = \dim_use:N \l_@@_line_width_dim
8084     }
8085   }
8086 }
8087 }
8088 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8089 {
8090   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8091   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8092   \@@_cut_on_hyphen:w #2 \q_stop
8093   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8094   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8095   \@@_cut_on_hyphen:w #3 \q_stop
8096   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8097   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8098   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8099   {
8100     \use:e
8101     {
8102       \@@_hline:n
8103       {
8104         position = ##1 ,
8105         start = \l_@@_tmpd_tl ,
8106         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8107         total-width = \dim_use:N \l_@@_line_width_dim
8108       }
8109     }
8110   }
8111 }

```

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).

```

8112 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8113 {
8114   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8115   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8116   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8117   { \@@_error:n { borders~forbidden } }
8118   {
8119     \tl_clear_new:N \l_@@_borders_tikz_tl
8120     \keys_set:no
8121     { nicematrix / OnlyForTikzInBorders }
8122     \l_@@_borders_clist
8123     \@@_cut_on_hyphen:w #2 \q_stop
8124     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8125     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8126     \@@_cut_on_hyphen:w #3 \q_stop
8127     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }

```

```

8128     \tl_set:Np \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8129     \@@_stroke_borders_block_i:
8130 }
8131 }

8132 \hook_gput_code:nnn { begindocument } { . }
8133 {
8134     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8135     {
8136         \c_@@_pgfortikzpicture_tl
8137         \@@_stroke_borders_block_ii:
8138         \c_@@_endpgfortikzpicture_tl
8139     }
8140 }

8141 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8142 {
8143     \pgfrememberpicturepositiononpagetrue
8144     \pgf@relevantforpicturesizefalse
8145     \CT@arc@
8146     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8147     \@@_clist_if_in:NnT \l_@@_borders_clist { right }
8148     { \@@_stroke_vertical:n \l_tmpb_tl }
8149     \@@_clist_if_in:NnT \l_@@_borders_clist { left }
8150     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8151     \@@_clist_if_in:NnT \l_@@_borders_clist { bottom }
8152     { \@@_stroke_horizontal:n \l_tmpa_tl }
8153     \@@_clist_if_in:NnT \l_@@_borders_clist { top }
8154     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8155 }

8156 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8157 {
8158     tikz .code:n =
8159         \cs_if_exist:NTF \tikzpicture
8160         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8161         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8162     tikz .value_required:n = true ,
8163     top .code:n = ,
8164     bottom .code:n = ,
8165     left .code:n = ,
8166     right .code:n = ,
8167     unknown .code:n = \@@_error:n { bad~border }
8168 }

```

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).

```

8169 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8170 {
8171     \@@_qpoint:n \l_@@_tmpc_tl
8172     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8173     \@@_qpoint:n \l_tmpa_tl
8174     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8175     \@@_qpoint:n { #1 }
8176     \tl_if_empty:NTF \l_@@_borders_tikz_tl
8177     {
8178         \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8179         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8180         \pgfusepathqstroke
8181     }
8182     {
8183         \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8184         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8185     }
8186 }

```

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).

```

8187 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8188 {
8189   \@@_qpoint:n \l_@@_tmpd_tl
8190   \@@_clist_if_in:NnTF \l_@@_borders_clist { left }
8191     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8192     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8193   \@@_qpoint:n \l_tmpb_tl
8194   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8195   \@@_qpoint:n { #1 }
8196   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8197     {
8198       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8199       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8200       \pgfusepathqstroke
8201     }
8202     {
8203       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8204         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8205     }
8206 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8207 \keys_define:nn { nicematrix / BlockBorders }
8208 {
8209   borders .clist_set:N = \l_@@_borders_clist ,
8210   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8211   rounded-corners .default:n = 4 pt ,
8212   line-width .dim_set:N = \l_@@_line_width_dim
8213 }

```

The following command will be used if the key tikz has been used for the command \Block.

#1 is a *list of lists* of Tikz keys used with the path.

Example: `{\offset=1pt,draw,red},{\offset=2pt,draw,blue}}`

which arises from a command such as :

`\Block[tikz={\offset=1pt,draw,red},tikz={\offset=2pt,draw,blue}]{2-2}{}`

The arguments #2 and #3 are the coordinates of the first cell and #4 and #5 the coordinates of the last cell of the block.

```

8214 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8215 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8216 {
8217   \begin { tikzpicture }
8218   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8219   \clist_map_inline:nn { #1 }
8220   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8221     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8222     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8223     (
8224       [
8225         xshift = \dim_use:N \l_@@_offset_dim ,
8226         yshift = - \dim_use:N \l_@@_offset_dim
8227       ]
8228       #2 -| #3
8229     )
8230     rectangle
8231     (
8232       [

```

```

8233         xshift = - \dim_use:N \l_@@_offset_dim ,
8234         yshift = \dim_use:N \l_@@_offset_dim
8235     ]
8236     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8237 ) ;
8238 }
8239 \end { tikzpicture }
8240 }

8241 \keys_define:nn { nicematrix / SpecialOffset }
8242 { offset .dim_set:N = \l_@@_offset_dim }

```

In some circumstances, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock`: which has the same syntax as the standard command `\Block` but which is no-op.

```

8243 \cs_new_protected:Npn \@@_NullBlock:
8244 { \@@_collect_options:n { \@@_NullBlock_i: } }
8245 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8246 { }

```

28 How to draw the dotted lines transparently

```

8247 \cs_set_protected:Npn \@@_renew_matrix:
8248 {
8249     \RenewDocumentEnvironment { pmatrix } { } {
8250         { \pNiceMatrix }
8251         { \endpNiceMatrix }
8252     }
8253     \RenewDocumentEnvironment { vmatrix } { } {
8254         { \vNiceMatrix }
8255         { \endvNiceMatrix }
8256     }
8257     \RenewDocumentEnvironment { Vmatrix } { } {
8258         { \VNiceMatrix }
8259         { \endVNiceMatrix }
8260     }
8261     \RenewDocumentEnvironment { bmatrix } { } {
8262         { \bNiceMatrix }
8263         { \endbNiceMatrix }
8264     }
8265     \RenewDocumentEnvironment { Bmatrix } { } {
8266         { \BNiceMatrix }
8267         { \endBNiceMatrix }
8268     }
8269 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8265 \keys_define:nn { nicematrix / Auto }
8266 {
8267     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8268     columns-type .value_required:n = true ,
8269     l .meta:n = { columns-type = l } ,
8270     r .meta:n = { columns-type = r } ,
8271     c .meta:n = { columns-type = c } ,
8272     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8273     delimiters / color .value_required:n = true ,
8274     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8275     delimiters / max-width .default:n = true ,
8276     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8277     delimiters .value_required:n = true ,

```

```

8278     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8279     rounded-corners .default:n = 4 pt
8280 }

8281 \NewDocumentCommand \AutoNiceMatrixWithDelims
8282 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8283 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8284 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8285 {

```

The group is for the protection of the keys.

```

8286     \group_begin:
8287     \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8288     \use:e
8289     {
8290         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8291         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8292         [ \exp_not:o \l_tmpa_tl ]
8293     }
8294     \int_if_zero:nT \l_@@_first_row_int
8295     {
8296         \int_if_zero:nT \l_@@_first_col_int { & }
8297         \prg_replicate:nn { #4 - 1 } { & }
8298         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8299     }
8300     \prg_replicate:nn { #3 }
8301     {
8302         \int_if_zero:nT \l_@@_first_col_int { & }

```

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).

```

8303         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8304         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8305     }
8306     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8307     {
8308         \int_if_zero:nT \l_@@_first_col_int { & }
8309         \prg_replicate:nn { #4 - 1 } { & }
8310         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8311     }
8312     \end { NiceArrayWithDelims }
8313     \group_end:
8314 }

8315 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8316 {
8317     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8318     {
8319         \bool_gset_true:N \g_@@_delims_bool
8320         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8321         \AutoNiceMatrixWithDelims { #2 } { #3 }
8322     }
8323 }

8324 \@@_define_com:nnn p ( )
8325 \@@_define_com:nnn b [ ]
8326 \@@_define_com:nnn v | |
8327 \@@_define_com:nnn V \l \l
8328 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8329 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8330 {

```

```

8331 \group_begin:
8332 \bool_gset_false:N \g_@@_delims_bool
8333 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8334 \group_end:
8335 }

```

30 The redefinition of the command `\dotfill`

```

8336 \cs_set_eq:NN \@@_old_dotfill \dotfill
8337 \cs_new_protected:Npn \@@_dotfill:
8338 {

```

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}`).

```

8339 \@@_old_dotfill
8340 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8341 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8342 \cs_new_protected:Npn \@@_dotfill_i:
8343 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

31 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.

```

8344 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8345 {
8346 \tl_gput_right:Ne \g_@@_pre_code_after_tl
8347 {
8348 \@@_actually_diagbox:nnnnnn
8349 { \int_use:N \c@iRow }
8350 { \int_use:N \c@jCol }
8351 { \int_use:N \c@iRow }
8352 { \int_use:N \c@jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

8353 { \g_@@_row_style_tl \exp_not:n { #1 } }
8354 { \g_@@_row_style_tl \exp_not:n { #2 } }
8355 }

```

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`.

```

8356 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8357 {
8358 { \int_use:N \c@iRow }
8359 { \int_use:N \c@jCol }
8360 { \int_use:N \c@iRow }
8361 { \int_use:N \c@jCol }

```


The last argument is for the name of the block.

```

8362     { }
8363   }
8364 }

```

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.

```

8365 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8366 {
8367   \pgfpicture
8368   \pgf@relevantforpicturesizefalse
8369   \pgfrememberpicturepositiononpagetrue
8370   \@@_qpoint:n { row - #1 }
8371   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8372   \@@_qpoint:n { col - #2 }
8373   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8374   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8375   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8376   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8377   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8378   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8379   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8380   {

```

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.

```

8381     \CT@arc@
8382     \pgfsetroundcap
8383     \pgfusepathqstroke
8384   }
8385   \pgfset { inner~sep = 1 pt }
8386   \pgfscope
8387   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8388   \pgfnode { rectangle } { south~west }
8389   {
8390     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8391     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8392     \end { minipage }
8393   }
8394   { }
8395   { }
8396 \endpgfscope
8397 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8398 \pgfnode { rectangle } { north~east }
8399 {
8400   \begin { minipage } { 20 cm }
8401   \raggedleft
8402   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8403   \end { minipage }
8404 }
8405 { }
8406 { }
8407 \endpgfpicture
8408 }

```

32 The keyword `\CodeAfter`

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. 84.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8409 \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 `\`.

```
8410 \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`.

```
8411 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8412 {
8413   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8414   \@@_CodeAfter_iv:n
8415 }
```

We catch the argument of the command `\end` (in `#1`).

```
8416 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8417 {
```

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.

```
8418   \str_if_eq:eeTF \@currenvir { #1 }
8419   { \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`.

```
8420   {
8421     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8422     \@@_CodeAfter_ii:n
8423   }
8424 }
```

33 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
8425 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8426 {
8427   \pgfpicture
8428   \pgfrememberpicturepositiononpagetrue
8429   \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.

```

8430 \@@_qpoint:n { row - 1 }
8431 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8432 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8433 \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).

```

8434 \bool_if:nTF { #3 }
8435 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8436 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8437 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8438 {
8439   \cs_if_exist:cT
8440   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8441   {
8442     \pgfpointanchor
8443     { \@@_env: - ##1 - #2 }
8444     { \bool_if:nTF { #3 } { west } { east } }
8445     \dim_set:Nn \l_tmpa_dim
8446     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8447   }
8448 }

```

Now we can put the delimiter with a node of PGF.

```

8449 \pgfset { inner~sep = \c_zero_dim }
8450 \dim_zero:N \nulldelimiterspace
8451 \pgftransformshift
8452 {
8453   \pgfpoint
8454   { \l_tmpa_dim }
8455   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8456 }
8457 \pgfnode
8458 { rectangle }
8459 { \bool_if:nTF { #3 } { east } { west } }
8460 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8461 \nullfont
8462 \c_math_toggle_token
8463 \@@_color:o \l_@@_delimiters_color_tl
8464 \bool_if:nTF { #3 } { \left #1 } { \left . }
8465 \vcenter
8466 {
8467   \nullfont
8468   \hrule \@height
8469   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8470   \@depth \c_zero_dim
8471   \@width \c_zero_dim
8472 }
8473 \bool_if:nTF { #3 } { \right . } { \right #1 }
8474 \c_math_toggle_token
8475 }
8476 { }
8477 { }
8478 \endpgfpicture
8479 }

```

34 The command \SubMatrix

```

8480 \keys_define:nn { nicematrix / sub-matrix }
8481 {
8482   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8483   extra-height .value_required:n = true ,
8484   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8485   left-xshift .value_required:n = true ,
8486   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8487   right-xshift .value_required:n = true ,
8488   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8489   xshift .value_required:n = true ,
8490   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8491   delimiters / color .value_required:n = true ,
8492   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8493   slim .default:n = true ,
8494   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8495   hlines .default:n = all ,
8496   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8497   vlines .default:n = all ,
8498   hvlines .meta:n = { hlines, vlines } ,
8499   hvlines .value_forbidden:n = true
8500 }
8501 \keys_define:nn { nicematrix }
8502 {
8503   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8504   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8505   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8506   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8507 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8508 \keys_define:nn { nicematrix / SubMatrix }
8509 {
8510   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8511   delimiters / color .value_required:n = true ,
8512   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8513   hlines .default:n = all ,
8514   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8515   vlines .default:n = all ,
8516   hvlines .meta:n = { hlines, vlines } ,
8517   hvlines .value_forbidden:n = true ,
8518   name .code:n =
8519     \tl_if_empty:nTF { #1 }
8520     { \@@_error:n { Invalid-name } }
8521     {
8522       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8523       {
8524         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8525         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8526         {
8527           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8528           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8529         }
8530       }
8531       { \@@_error:n { Invalid-name } }
8532     } ,
8533   name .value_required:n = true ,
8534   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8535   rules .value_required:n = true ,
8536   code .tl_set:N = \l_@@_code_tl ,

```

```

8537     code .value_required:n = true ,
8538     unknown .code:n = \@_error:n { Unknown~key~for~SubMatrix }
8539 }

8540 \NewDocumentCommand \@_SubMatrix_in_code_before { m m m m ! 0 { } }
8541 {
8542     \peek_remove_spaces:n
8543     {
8544         \tl_gput_right:Ne \g_@@_pre_code_after_tl
8545         {
8546             \SubMatrix { #1 } { #2 } { #3 } { #4 }
8547             [
8548                 delimiters / color = \l_@@_delimiters_color_tl ,
8549                 hlines = \l_@@_submatrix_hlines_clist ,
8550                 vlines = \l_@@_submatrix_vlines_clist ,
8551                 extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8552                 left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8553                 right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8554                 slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8555                 #5
8556             ]
8557         }
8558         \@_SubMatrix_in_code_before_i { #2 } { #3 }
8559     }
8560 }

8561 \NewDocumentCommand \@_SubMatrix_in_code_before_i
8562 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8563 { \@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8564 \cs_new_protected:Npn \@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8565 {
8566     \seq_gput_right:Ne \g_@@_submatrix_seq
8567     {
We use \str_if_eq:eeTF because it is fully expandable (and slightly faster than \tl_if_eq:nnTF).
8568         { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8569         { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8570         { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8571         { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8572     }
8573 }

```

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.

```

8574 \hook_gput_code:nnn { begindocument } { . }
8575 {
8576     \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8577     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

8578 \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8579 {
8580   \peek_remove_spaces:n
8581   {
8582     \@@_sub_matrix:nnnnnnn
8583     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8584   }
8585 }
8586 }

```

The following macro will compute $\backslash l_@@_first_i_tl$, $\backslash l_@@_first_j_tl$, $\backslash l_@@_last_i_tl$ and $\backslash l_@@_last_j_tl$ from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

8587 \NewDocumentCommand \@@_compute_i_j:nn
8588 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8589 { \@@_compute_i_j:nnnn #1 #2 }

8590 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8591 {
8592   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8593   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8594   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8595   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8596   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8597     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8598   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8599     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8600   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8601     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8602   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8603     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8604 }

8605 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8606 {
8607   \group_begin:

```

The four following token lists correspond to the position of the \backslash SubMatrix.

```

8608 \@@_compute_i_j:nn { #2 } { #3 }
8609 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8610   { \cs_set_nopar:Npn \arraystretch { 1 } }
8611 \bool_lazy_or:nnTF
8612   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8613   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8614   { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8615   {
8616     \str_clear_new:N \l_@@_submatrix_name_str
8617     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8618     \pgfpicture
8619     \pgfrememberpicturepositiononpagetrue
8620     \pgf@relevantforpicturesizefalse
8621     \pgfset { inner~sep = \c_zero_dim }
8622     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8623     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \backslash int_step_inline:nnn is provided by curriification.

```

8624 \bool_if:NTF \l_@@_submatrix_slim_bool
8625   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8626   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8627   {
8628     \cs_if_exist:cT
8629       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8630       {
8631         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8632         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim

```

```

8633         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8634     }
8635     \cs_if_exist:cT
8636     { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8637     {
8638         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8639         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8640         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8641     }
8642 }
8643 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8644 { \@@_error:nn { Impossible~delimiter } { left } }
8645 {
8646     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8647     { \@@_error:nn { Impossible~delimiter } { right } }
8648     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8649 }
8650 \endpgfpicture
8651 }
8652 \group_end:
8653 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8654 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8655 {
8656     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8657     \dim_set:Nn \l_@@_y_initial_dim
8658     {
8659         \fp_to_dim:n
8660         {
8661             \pgf@y
8662             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8663         }
8664     }
8665     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8666     \dim_set:Nn \l_@@_y_final_dim
8667     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8668     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8669     {
8670         \cs_if_exist:cT
8671         { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8672         {
8673             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8674             \dim_set:Nn \l_@@_y_initial_dim
8675             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8676         }
8677         \cs_if_exist:cT
8678         { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8679         {
8680             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8681             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8682             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8683         }
8684     }
8685     \dim_set:Nn \l_tmpa_dim
8686     {
8687         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8688         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8689     }
8690     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8691 \group_begin:
8692 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8693 \@@_set_CT@arc@:o \l_@@_rules_color_tl
8694 \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`.

```

8695 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8696 {
8697   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8698   {
8699     \int_compare:nNnT
8700       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8701     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8702       \@@_qpoint:n { col - ##1 }
8703       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8704       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8705       \pgfusepathqstroke
8706     }
8707   }
8708 }

```

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.

```

8709 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8710 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8711 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8712 {
8713   \bool_lazy_and:nnTF
8714     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8715     {
8716       \int_compare_p:nNn
8717         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8718       {
8719         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8720         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8721         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8722         \pgfusepathqstroke
8723       }
8724       { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8725     }

```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

8726 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8727 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8728 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8729 {
8730   \bool_lazy_and:nnTF
8731     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8732     {
8733       \int_compare_p:nNn
8734         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8735       {
8736         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8737 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8738 \dim_set:Nn \l_tmpa_dim

```



```

8739         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8740 \str_case:nn { #1 }
8741 {
8742     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8743     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8744     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8745     }
8746 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8747 \dim_set:Nn \l_tmpb_dim
8748 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8749 \str_case:nn { #2 }
8750 {
8751     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8752     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8753     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8754 }
8755 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8756 \pgfusepathqstroke
8757 \group_end:
8758 }
8759 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8760 }

```

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).

```

8761 \str_if_empty:NF \l_@@_submatrix_name_str
8762 {
8763     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8764     \l_@@_x_initial_dim \l_@@_y_initial_dim
8765     \l_@@_x_final_dim \l_@@_y_final_dim
8766 }
8767 \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`.

```

8768 \begin { pgfscope }
8769 \pgftransformshift
8770 {
8771     \pgfpoint
8772     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8773     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8774 }
8775 \str_if_empty:NTF \l_@@_submatrix_name_str
8776 { \@@_node_left:nn #1 { } }
8777 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8778 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8779 \pgftransformshift
8780 {
8781     \pgfpoint
8782     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8783     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8784 }
8785 \str_if_empty:NTF \l_@@_submatrix_name_str
8786 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8787 {
8788     \@@_node_right:nnnn #2
8789     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8790 }

```

```

8791 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8792 \flag_clear_new:N \l_@@_code_flag
8793 \l_@@_code_tl
8794 }

```

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`.

```

8795 \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.

```

8796 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8797 {
8798   \use:e
8799   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8800 }

```

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 -.

```

8801 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8802 { #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.

```

8803 \tl_const:Nn \c_@@_integers_alist_tl
8804 {
8805   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8806   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8807   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8808   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8809 }

```

```

8810 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8811 {

```

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`.

```

8812 \tl_if_empty:nTF { #2 }
8813 {
8814   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8815   {
8816     \flag_raise:N \l_@@_code_flag
8817     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8818     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8819     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8820   }
8821   { #1 }
8822 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8823     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8824 }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

8825 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8826 {
8827   \str_case:nnF { #1 }
8828   {
8829     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8830     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8831   }

```

Now the case of a node of the form $i-j$.

```

8832   {
8833     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8834     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8835   }
8836 }

```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

8837 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8838 {
8839   \pgfnode
8840   { rectangle }
8841   { east }
8842   {
8843     \nullfont
8844     \c_math_toggle_token
8845     \@@_color:o \l_@@_delimiters_color_tl
8846     \left #1
8847     \vcenter
8848     {
8849       \nullfont
8850       \hrule \@height \l_tmpa_dim
8851       \@depth \c_zero_dim
8852       \@width \c_zero_dim
8853     }
8854     \right .
8855     \c_math_toggle_token
8856   }
8857   { #2 }
8858   { }
8859 }

```

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.

```

8860 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8861 {
8862   \pgfnode
8863   { rectangle }
8864   { west }
8865   {
8866     \nullfont
8867     \c_math_toggle_token
8868     \colorlet { current-color } { . }
8869     \@@_color:o \l_@@_delimiters_color_tl
8870     \left .

```

```

8871 \vcenter
8872 {
8873   \nullfont
8874   \hrule \@height \l_tmpa_dim
8875         \@depth \c_zero_dim
8876         \@width \c_zero_dim
8877 }
8878 \right #1
8879 \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8880 ~ { \color { current-color } \smash { #4 } }
8881 \c_math_toggle_token
8882 }
8883 { #2 }
8884 { }
8885 }

```

35 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

8886 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8887 {
8888   \peek_remove_spaces:n
8889   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8890 }
8891 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8892 {
8893   \peek_remove_spaces:n
8894   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8895 }
8896 \keys_define:nn { nicematrix / Brace }
8897 {
8898   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8899   left-shorten .default:n = true ,
8900   left-shorten .value_forbidden:n = true ,
8901   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8902   right-shorten .default:n = true ,
8903   right-shorten .value_forbidden:n = true ,
8904   shorten .meta:n = { left-shorten , right-shorten } ,
8905   shorten .value_forbidden:n = true ,
8906   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8907   yshift .value_required:n = true ,
8908   yshift .initial:n = \c_zero_dim ,
8909   color .tl_set:N = \l_tmpa_tl ,
8910   color .value_required:n = true ,
8911   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8912 }

```

#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*.

```

8913 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8914 {
8915   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8916 \@@_compute_i_j:nn { #1 } { #2 }
8917 \bool_lazy_or:nnTF

```

```

8918 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8919 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8920 {
8921   \str_if_eq:eeTF { #5 } { under }
8922   { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8923   { \@@_error:nn { Construct~too~large } { \OverBrace } }
8924 }
8925 {
8926   \tl_clear:N \l_tmpa_tl
8927   \keys_set:nn { nicematrix / Brace } { #4 }
8928   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8929   \pgfpicture
8930   \pgfrememberpicturepositiononpagetrue
8931   \pgf@relevantforpicturesizefalse
8932   \bool_if:NT \l_@@_brace_left_shorten_bool
8933   {
8934     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8935     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8936     {
8937       \cs_if_exist:cT
8938       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8939       {
8940         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8941
8942         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8943         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8944       }
8945     }
8946   }
8947   \bool_lazy_or:nnT
8948   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8949   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8950   {
8951     \@@_qpoint:n { col - \l_@@_first_j_tl }
8952     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8953   }
8954   \bool_if:NT \l_@@_brace_right_shorten_bool
8955   {
8956     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8957     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8958     {
8959       \cs_if_exist:cT
8960       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8961       {
8962         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8963         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8964         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8965       }
8966     }
8967   }
8968   \bool_lazy_or:nnT
8969   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8970   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8971   {
8972     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8973     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8974   }
8975   \pgfset { inner~sep = \c_zero_dim }
8976   \str_if_eq:eeTF { #5 } { under }
8977   { \@@_underbrace_i:n { #3 } }
8978   { \@@_overbrace_i:n { #3 } }
8979   \endpgfpicture
8980 }

```

```

8981 \group_end:
8982 }

```

The argument is the text to put above the brace.

```

8983 \cs_new_protected:Npn \@@_overbrace_i:n #1
8984 {
8985   \@@_qpoint:n { row - \l_@@_first_i_tl }
8986   \pgftransformshift
8987   {
8988     \pgfpoint
8989     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8990     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8991   }
8992   \pgfnode
8993   { rectangle }
8994   { south }
8995   {
8996     \vtop
8997     {
8998       \group_begin:
8999       \everycr { }
9000       \halign
9001       {
9002         \hfil ## \hfil \crrc
9003         \@@_math_toggle: #1 \@@_math_toggle: \cr
9004         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9005         \c_math_toggle_token
9006         \overbrace
9007         {
9008           \hbox_to_wd:nn
9009           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9010           { }
9011         }
9012         \c_math_toggle_token
9013         \cr
9014       }
9015       \group_end:
9016     }
9017   }
9018   { }
9019   { }
9020 }

```

The argument is the text to put under the brace.

```

9021 \cs_new_protected:Npn \@@_underbrace_i:n #1
9022 {
9023   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9024   \pgftransformshift
9025   {
9026     \pgfpoint
9027     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9028     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9029   }
9030   \pgfnode
9031   { rectangle }
9032   { north }
9033   {
9034     \group_begin:
9035     \everycr { }
9036     \vbox
9037     {
9038       \halign
9039       {

```

```

9040         \hfil ## \hfil \crrc
9041         \c_math_toggle_token
9042         \underbrace
9043         {
9044             \hbox_to_wd:nn
9045             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9046             { }
9047         }
9048         \c_math_toggle_token
9049         \cr
9050         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9051         \@@_math_toggle: #1 \@@_math_toggle: \cr
9052     }
9053 }
9054 \group_end:
9055 }
9056 { }
9057 { }
9058 }

```

36 The command TikzEveryCell

```

9059 \bool_new:N \l_@@_not_empty_bool
9060 \bool_new:N \l_@@_empty_bool
9061
9062 \keys_define:nn { nicematrix / TikzEveryCell }
9063 {
9064     not-empty .code:n =
9065         \bool_lazy_or:nnTF
9066         \l_@@_in_code_after_bool
9067         \g_@@_recreate_cell_nodes_bool
9068         { \bool_set_true:N \l_@@_not_empty_bool }
9069         { \@@_error:n { detection-of-empty-cells } } ,
9070     not-empty .value_forbidden:n = true ,
9071     empty .code:n =
9072         \bool_lazy_or:nnTF
9073         \l_@@_in_code_after_bool
9074         \g_@@_recreate_cell_nodes_bool
9075         { \bool_set_true:N \l_@@_empty_bool }
9076         { \@@_error:n { detection-of-empty-cells } } ,
9077     empty .value_forbidden:n = true ,
9078     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9079 }
9080
9081
9082 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9083 {
9084     \IfPackageLoadedTF { tikz }
9085     {
9086         \group_begin:
9087         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

9088         \tl_set:Nn \l_tmpa_tl { { #2 } }
9089         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9090         { \@@_for_a_block:nnnnn ##1 }
9091         \@@_all_the_cells:
9092         \group_end:
9093     }

```

```

9094     { \@@_error:n { TikzEveryCell~without~tikz } }
9095   }
9096
9097   \tl_new:N \@@_i_tl
9098   \tl_new:N \@@_j_tl
9099
9100
9101   \cs_new_protected:Nn \@@_all_the_cells:
9102   {
9103     \int_step_variable:nNn \c@iRow \@@_i_tl
9104     {
9105       \int_step_variable:nNn \c@jCol \@@_j_tl
9106       {
9107         \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9108         {
9109           \@@_clist_if_in:Nf \l_@@_corners_cells_clist
9110           { \@@_i_tl - \@@_j_tl }
9111           {
9112             \bool_set_false:N \l_tmpa_bool
9113             \cs_if_exist:cTF
9114             { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9115             {
9116               \bool_if:NF \l_@@_empty_bool
9117               { \bool_set_true:N \l_tmpa_bool }
9118             }
9119             {
9120               \bool_if:NF \l_@@_not_empty_bool
9121               { \bool_set_true:N \l_tmpa_bool }
9122             }
9123             \bool_if:NT \l_tmpa_bool
9124             {
9125               \@@_block_tikz:nnnn
9126               \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9127             }
9128           }
9129         }
9130       }
9131     }
9132   }
9133
9134   \cs_new_protected:Nn \@@_for_a_block:nnnnn
9135   {
9136     \bool_if:NF \l_@@_empty_bool
9137     {
9138       \@@_block_tikz:nnnnn
9139       \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9140     }
9141     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9142   }
9143
9144   \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9145   {
9146     \int_step_inline:nnn { #1 } { #3 }
9147     {
9148       \int_step_inline:nnn { #2 } { #4 }
9149       { \cs_set:cpn { cell - ##1 - ####1 } { } }
9150     }
9151   }

```

37 The command \ShowCellNames

```

9152 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }

```



```

9153 {
9154   \dim_gzero_new:N \g_@@_tmpc_dim
9155   \dim_gzero_new:N \g_@@_tmpd_dim
9156   \dim_gzero_new:N \g_@@_tmpe_dim
9157   \int_step_inline:nn \c@iRow
9158   {
9159     \begin { pgfpicture }
9160     \@@_qpoint:n { row - ##1 }
9161     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9162     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9163     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9164     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9165     \bool_if:NTF \l_@@_in_code_after_bool
9166     \end { pgfpicture }
9167     \int_step_inline:nn \c@jCol
9168     {
9169       \hbox_set:Nn \l_tmpa_box
9170       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
9171       \begin { pgfpicture }
9172       \@@_qpoint:n { col - #####1 }
9173       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9174       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9175       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9176       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9177       \endpgfpicture
9178       \end { pgfpicture }
9179       \fp_set:Nn \l_tmpa_fp
9180       {
9181         \fp_min:nn
9182         {
9183           \fp_min:nn
9184           {
9185             \dim_ratio:nn
9186             { \g_@@_tmpd_dim }
9187             { \box_wd:N \l_tmpa_box }
9188           }
9189           {
9190             \dim_ratio:nn
9191             { \g_tmpb_dim }
9192             { \box_ht_plus_dp:N \l_tmpa_box }
9193           }
9194         }
9195         { 1.0 }
9196       }
9197       \box_scale:Nnn \l_tmpa_box
9198       { \fp_use:N \l_tmpa_fp }
9199       { \fp_use:N \l_tmpa_fp }
9200       \pgfpicture
9201       \pgfrememberpicturepositiononpagetrue
9202       \pgf@relevantforpicturesizefalse
9203       \pgftransformshift
9204       {
9205         \pgfpoint
9206         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9207         { \dim_use:N \g_tmpa_dim }
9208       }
9209       \pgfnode
9210       { rectangle }
9211       { center }
9212       { \box_use:N \l_tmpa_box }
9213       { }
9214       { }
9215       \endpgfpicture

```

```

9216     }
9217 }
9218 }
9219 \NewDocumentCommand \@@_ShowCellNames { }
9220 {
9221   \bool_if:NT \l_@@_in_code_after_bool
9222   {
9223     \pgfpicture
9224     \pgfrememberpicturepositiononpagetrue
9225     \pgf@relevantforpicturesizefalse
9226     \pgfpathrectanglecorners
9227     { \@@_qpoint:n { 1 } }
9228     {
9229       \@@_qpoint:n
9230       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9231     }
9232     \pgfsetfillopacity { 0.75 }
9233     \pgfsetfillcolor { white }
9234     \pgfusepathqfill
9235     \endpgfpicture
9236   }
9237   \dim_gzero_new:N \g_@@_tmpc_dim
9238   \dim_gzero_new:N \g_@@_tmpd_dim
9239   \dim_gzero_new:N \g_@@_tmpe_dim
9240   \int_step_inline:nn \c@iRow
9241   {
9242     \bool_if:NTF \l_@@_in_code_after_bool
9243     {
9244       \pgfpicture
9245       \pgfrememberpicturepositiononpagetrue
9246       \pgf@relevantforpicturesizefalse
9247     }
9248     { \begin { pgfpicture } }
9249     \@@_qpoint:n { row - ##1 }
9250     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9251     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9252     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9253     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9254     \bool_if:NTF \l_@@_in_code_after_bool
9255     { \endpgfpicture }
9256     { \end { pgfpicture } }
9257     \int_step_inline:nn \c@jCol
9258     {
9259       \hbox_set:Nn \l_tmpa_box
9260       {
9261         \normalfont \Large \sffamily \bfseries
9262         \bool_if:NTF \l_@@_in_code_after_bool
9263         { \color { red } }
9264         { \color { red ! 50 } }
9265         ##1 - ####1
9266       }
9267       \bool_if:NTF \l_@@_in_code_after_bool
9268       {
9269         \pgfpicture
9270         \pgfrememberpicturepositiononpagetrue
9271         \pgf@relevantforpicturesizefalse
9272       }
9273       { \begin { pgfpicture } }
9274       \@@_qpoint:n { col - ####1 }
9275       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9276       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9277       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9278       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

9279 \bool_if:NTF \l_@@_in_code_after_bool
9280 { \endpgfpicture }
9281 { \end { pgfpicture } }
9282 \fp_set:Nn \l_tmpa_fp
9283 {
9284   \fp_min:nn
9285   {
9286     \fp_min:nn
9287     { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9288     { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9289   }
9290   { 1.0 }
9291 }
9292 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9293 \pgfpicture
9294 \pgfrememberpicturepositiononpagetrue
9295 \pgf@relevantforpicturesizefalse
9296 \pgftransformshift
9297 {
9298   \pgfpoint
9299   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9300   { \dim_use:N \g_tmpa_dim }
9301 }
9302 \pgfnode
9303 { rectangle }
9304 { center }
9305 { \box_use:N \l_tmpa_box }
9306 { }
9307 { }
9308 \endpgfpicture
9309 }
9310 }
9311 }

```

38 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.

```

9312 \bool_new:N \g_@@_footnotehyper_bool

```

The boolean `\g_@@_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.

```

9313 \bool_new:N \g_@@_footnote_bool
9314 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9315 {
9316   The~key~'\l_keys_key_str'~is~unknown. \\
9317   That~key~will~be~ignored. \\
9318   For~a~list~of~the~available~keys,~type~H~<return>.
9319 }
9320 {
9321   The~available~keys~are~(in~alphabetic~order):~
9322   footnote,~
9323   footnotehyper,~
9324   messages-for-Overleaf,~
9325   no-test-for-array,~
9326   renew-dots,~and~

```

```

9327     renew-matrix.
9328 }
9329 \keys_define:nn { nicematrix / Package }
9330 {
9331     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9332     renew-dots .value_forbidden:n = true ,
9333     renew-matrix .code:n = \@@_renew_matrix: ,
9334     renew-matrix .value_forbidden:n = true ,
9335     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9336     footnote .bool_set:N = \g_@@_footnote_bool ,
9337     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9338     no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
9339     no-test-for-array .default:n = true ,
9340     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9341 }
9342 \ProcessKeysOptions { nicematrix / Package }

9343 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9344 {
9345     You~can't~use~the~option~'footnote'~because~the~package~
9346     footnotehyper~has~already~been~loaded.~
9347     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9348     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9349     of~the~package~footnotehyper.\\
9350     The~package~footnote~won't~be~loaded.
9351 }
9352 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9353 {
9354     You~can't~use~the~option~'footnotehyper'~because~the~package~
9355     footnote~has~already~been~loaded.~
9356     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9357     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9358     of~the~package~footnote.\\
9359     The~package~footnotehyper~won't~be~loaded.
9360 }

9361 \bool_if:NT \g_@@_footnote_bool
9362 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9363     \IfClassLoadedTF { beamer }
9364     { \bool_set_false:N \g_@@_footnote_bool }
9365     {
9366         \IfPackageLoadedTF { footnotehyper }
9367         { \@@_error:n { footnote~with~footnotehyper~package } }
9368         { \usepackage { footnote } }
9369     }
9370 }
9371 \bool_if:NT \g_@@_footnotehyper_bool
9372 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9373     \IfClassLoadedTF { beamer }
9374     { \bool_set_false:N \g_@@_footnote_bool }
9375     {
9376         \IfPackageLoadedTF { footnote }
9377         { \@@_error:n { footnotehyper~with~footnote~package } }
9378         { \usepackage { footnotehyper } }
9379     }
9380     \bool_set_true:N \g_@@_footnote_bool
9381 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

39 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.

```

9382 \bool_new:N \l_@@_underscore_loaded_bool
9383 \IfPackageLoadedT { underscore }
9384 { \bool_set_true:N \l_@@_underscore_loaded_bool }

9385 \hook_gput_code:nnn { begindocument } { . }
9386 {
9387   \bool_if:NF \l_@@_underscore_loaded_bool
9388   {
9389     \IfPackageLoadedT { underscore }
9390     { \@@_error:n { underscore-after-nicematrix } }
9391   }
9392 }
```

40 Error messages of the package

```

9393 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9394 { \str_const:Nn \c_@@_available_keys_str { } }
9395 {
9396   \str_const:Nn \c_@@_available_keys_str
9397   { For~a~list~of~the~available~keys,~type~H~<return>. }
9398 }

9399 \seq_new:N \g_@@_types_of_matrix_seq
9400 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9401 {
9402   NiceMatrix ,
9403   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9404 }
9405 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9406 { \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:NoF` 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`.

```

9407 \cs_new_protected:Npn \@@_error_too_much_cols:
9408 {
9409   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9410   { \@@_fatal:nn { too-much-cols-for-array } }
9411   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9412   { \@@_fatal:n { too-much-cols-for-matrix } }
9413   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9414   { \@@_fatal:n { too-much-cols-for-matrix } }
9415   \bool_if:NF \l_@@_last_col_without_value_bool
9416   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9417 }
```

The following command must *not* be protected since it's used in an error message.

```

9418 \cs_new:Npn \@@_message_hdotsfor:
9419 {
9420   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9421   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9422 }
9423 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9424 {
9425   Incompatible~options.\\
9426   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9427   The~output~will~not~be~reliable.
9428 }
9429 \@@_msg_new:nn { negative~weight }
9430 {
9431   Negative~weight.\\
9432   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9433   the~value~'\int_use:N \l_@@_weight_int'.\\
9434   The~absolute~value~will~be~used.
9435 }
9436 \@@_msg_new:nn { last~col~not~used }
9437 {
9438   Column~not~used.\\
9439   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9440   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9441 }
9442 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9443 {
9444   Too~much~columns.\\
9445   In~the~row~\int_eval:n { \c@iRow },~
9446   you~try~to~use~more~columns~
9447   than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
9448   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9449   (plus~the~exterior~columns).~This~error~is~fatal.
9450 }
9451 \@@_msg_new:nn { too~much~cols~for~matrix }
9452 {
9453   Too~much~columns.\\
9454   In~the~row~\int_eval:n { \c@iRow },~
9455   you~try~to~use~more~columns~than~allowed~by~your~
9456   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
9457   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9458   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9459   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9460   \token_to_str:N \setcounter\ to~change~that~value).~
9461   This~error~is~fatal.
9462 }
9463 \@@_msg_new:nn { too~much~cols~for~array }
9464 {
9465   Too~much~columns.\\
9466   In~the~row~\int_eval:n { \c@iRow },~
9467   ~you~try~to~use~more~columns~than~allowed~by~your~
9468   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9469   \int_use:N \g_@@_static_num_of_col_int\
9470   ~(plus~the~potential~exterior~ones).~
9471   This~error~is~fatal.
9472 }
9473 \@@_msg_new:nn { columns~not~used }
9474 {
9475   Columns~not~used.\\
9476   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9477   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\

```

```

9478     The~columns~you~did~not~used~won't~be~created.\\
9479     You~won't~have~similar~error~message~till~the~end~of~the~document.
9480 }

9481 \@@_msg_new:nn { empty~preamble }
9482 {
9483     Empty~preamble.\\
9484     The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9485     This~error~is~fatal.
9486 }

9487 \@@_msg_new:nn { in~first~col }
9488 {
9489     Erroneous~use.\\
9490     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9491     That~command~will~be~ignored.
9492 }

9493 \@@_msg_new:nn { in~last~col }
9494 {
9495     Erroneous~use.\\
9496     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9497     That~command~will~be~ignored.
9498 }

9499 \@@_msg_new:nn { in~first~row }
9500 {
9501     Erroneous~use.\\
9502     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9503     That~command~will~be~ignored.
9504 }

9505 \@@_msg_new:nn { in~last~row }
9506 {
9507     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9508     That~command~will~be~ignored.
9509 }

9510 \@@_msg_new:nn { caption~outside~float }
9511 {
9512     Key~caption~forbidden.\\
9513     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9514     environment.~This~key~will~be~ignored.
9515 }

9516 \@@_msg_new:nn { short~caption~without~caption }
9517 {
9518     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9519     However,~your~'short~caption'~will~be~used~as~'caption'.
9520 }

9521 \@@_msg_new:nn { double~closing~delimiter }
9522 {
9523     Double~delimiter.\\
9524     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9525     delimiter.~This~delimiter~will~be~ignored.
9526 }

9527 \@@_msg_new:nn { delimiter~after~opening }
9528 {
9529     Double~delimiter.\\
9530     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9531     delimiter.~That~delimiter~will~be~ignored.
9532 }

9533 \@@_msg_new:nn { bad~option~for~line~style }
9534 {
9535     Bad~line~style.\\
9536     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~

```

```

9537     is~'standard'.~That~key~will~be~ignored.
9538 }
9539 \@@_msg_new:nn { Identical~notes~in~caption }
9540 {
9541     Identical~tabular~notes.\\
9542     You~can't~put~several~notes~with~the~same~content~in~
9543     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9544     If~you~go~on,~the~output~will~probably~be~erroneous.
9545 }
9546 \@@_msg_new:nn { tabularnote~below~the~tabular }
9547 {
9548     \token_to_str:N \tabularnote\ forbidden\\
9549     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9550     of~your~tabular~because~the~caption~will~be~composed~below~
9551     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9552     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9553     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9554     no~similar~error~will~raised~in~this~document.
9555 }
9556 \@@_msg_new:nn { Unknown~key~for~rules }
9557 {
9558     Unknown~key.\\
9559     There~is~only~two~keys~available~here:~width~and~color.\\
9560     Your~key~'\l_keys_key_str'~will~be~ignored.
9561 }
9562 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9563 {
9564     Unknown~key.\\
9565     There~is~only~two~keys~available~here:~
9566     'empty'~and~'not~empty'.\\
9567     Your~key~'\l_keys_key_str'~will~be~ignored.
9568 }
9569 \@@_msg_new:nn { Unknown~key~for~rotate }
9570 {
9571     Unknown~key.\\
9572     The~only~key~available~here~is~'c'.\\
9573     Your~key~'\l_keys_key_str'~will~be~ignored.
9574 }
9575 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9576 {
9577     Unknown~key.\\
9578     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9579     It~you~go~on,~you~will~probably~have~other~errors. \\
9580     \c_@@_available_keys_str
9581 }
9582 {
9583     The~available~keys~are~(in~alphabetic~order):~
9584     ccommand,~
9585     color,~
9586     command,~
9587     dotted,~
9588     letter,~
9589     multiplicity,~
9590     sep-color,~
9591     tikz,~and~total~width.
9592 }
9593 \@@_msg_new:nnn { Unknown~key~for~xdots }
9594 {
9595     Unknown~key.\\
9596     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9597     \c_@@_available_keys_str

```



```

9598 }
9599 {
9600   The-available-keys-are-(in-alphabetic-order):~
9601   'color',~
9602   'horizontal-labels',~
9603   'inter',~
9604   'line-style',~
9605   'radius',~
9606   'shorten',~
9607   'shorten-end'~and~'shorten-start'.
9608 }
9609 \@@_msg_new:nn { Unknown-key-for-rowcolors }
9610 {
9611   Unknown-key.\\
9612   As-for-now,~there-is-only~two-keys~available~here:~'cols'~and~'respect-blocks'~
9613   (and-you-try-to-use~'\l_keys_key_str')\\
9614   That-key-will-be-ignored.
9615 }
9616 \@@_msg_new:nn { label-without-caption }
9617 {
9618   You-can't-use-the-key~'label'~in-your~'{NiceTabular}'~because~
9619   you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9620 }
9621 \@@_msg_new:nn { W-warning }
9622 {
9623   Line~\msg_line_number:~The-cell-is-too-wide-for-your-column~'W'~
9624   (row~\int_use:N \c@iRow).
9625 }
9626 \@@_msg_new:nn { Construct-too-large }
9627 {
9628   Construct-too-large.\\
9629   Your-command~\token_to_str:N #1
9630   can't-be-drawn-because-your-matrix-is-too-small.\\
9631   That-command-will-be-ignored.
9632 }
9633 \@@_msg_new:nn { underscore-after-nicematrix }
9634 {
9635   Problem-with~'underscore'.\\
9636   The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9637   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
9638   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9639 }
9640 \@@_msg_new:nn { ampersand-in-light-syntax }
9641 {
9642   Ampersand-forbidden.\\
9643   You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
9644   ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9645 }
9646 \@@_msg_new:nn { double-backslash-in-light-syntax }
9647 {
9648   Double-backslash-forbidden.\\
9649   You-can't-use~\token_to_str:N
9650   \\~to-separate-rows-because-the-key~'light-syntax'~
9651   is-in-force.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
9652   (set-by-the-key~'end-of-row').~This-error-is-fatal.
9653 }
9654 \@@_msg_new:nn { hlines-with-color }
9655 {
9656   Incompatible-keys.\\
9657   You-can't-use-the-keys~'hlines',~'vlines'~or~'hvlines'~for-a~

```

```

9658     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9659     However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9660     Your~key~will~be~discarded.
9661 }

9662 \@@_msg_new:nn { bad-value-for-baseline }
9663 {
9664     Bad-value-for-baseline.\\
9665     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9666     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9667     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9668     the~form~'line-i'.\\
9669     A~value~of~1~will~be~used.
9670 }

9671 \@@_msg_new:nn { detection-of-empty-cells }
9672 {
9673     Problem-with~'not-empty'\\
9674     For~technical~reasons,~you~must~activate~
9675     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9676     in~order~to~use~the~key~'\l_keys_key_str'.\\
9677     That~key~will~be~ignored.
9678 }

9679 \@@_msg_new:nn { siunitx-not-loaded }
9680 {
9681     siunitx-not-loaded\\
9682     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9683     That~error~is~fatal.
9684 }

9685 \@@_msg_new:nn { ragged2e-not-loaded }
9686 {
9687     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9688     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:o
9689     \l_keys_key_str'~will~be~used~instead.
9690 }

9691 \@@_msg_new:nn { Invalid-name }
9692 {
9693     Invalid-name.\\
9694     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9695     \SubMatrix\ of~your~\@@_full_name_env:.\\
9696     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9697     This~key~will~be~ignored.
9698 }

9699 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9700 {
9701     Wrong-line.\\
9702     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9703     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9704     number~is~not~valid.~It~will~be~ignored.
9705 }

9706 \@@_msg_new:nn { Impossible-delimiter }
9707 {
9708     Impossible-delimiter.\\
9709     It's~impossible~to~draw~the~#1~delimiter~of~your~
9710     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9711     in~that~column.
9712     \bool_if:NT \l_@@_submatrix_slim_bool
9713     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9714     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9715 }

9716 \@@_msg_new:nnn { width-without-X-columns }
9717 {

```

```

9718     You-have-used-the-key~'width'~but-you-have-put~no~'X'~column.~
9719     That-key-will-be-ignored.
9720 }
9721 {
9722     This-message-is~the-message~'width~without~X~columns'~
9723     of~the~module~'nicematrix'.~
9724     The-experimented-users-can-disable-that-message-with~
9725     \token_to_str:N \msg_redirect_name:nnn.\\
9726 }
9727
9728 \@@_msg_new:nn { key-multiplicity-with-dotted }
9729 {
9730     Incompatible-keys. \\
9731     You-have-used-the-key~'multiplicity'~with~the-key~'dotted'~
9732     in~a~'custom-line'.~They-are-incompatible. \\
9733     The-key~'multiplicity'~will-be-discarded.
9734 }
9735 \@@_msg_new:nn { empty-environment }
9736 {
9737     Empty-environment.\\
9738     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9739 }
9740 \@@_msg_new:nn { No-letter-and-no-command }
9741 {
9742     Erroneous-use.\\
9743     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9744     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9745     ~'ccommand'~(to~draw~horizontal~rules).\\
9746     However,~you~can~go~on.
9747 }
9748 \@@_msg_new:nn { Forbidden-letter }
9749 {
9750     Forbidden-letter.\\
9751     You-can't-use-the-letter~'#1'~for~a~customized-line.\\
9752     It-will-be-ignored.
9753 }
9754 \@@_msg_new:nn { Several-letters }
9755 {
9756     Wrong-name.\\
9757     You-must-use-only-one-letter-as-value-for-the-key~'letter'~(and-you~
9758     have-used~'\l_@@_letter_str').\\
9759     It-will-be-ignored.
9760 }
9761 \@@_msg_new:nn { Delimiter-with-small }
9762 {
9763     Delimiter-forbidden.\\
9764     You-can't-put-a-delimiter~in~the-preamble~of~your~\@@_full_name_env:\
9765     because~the~key~'small'~is~in~force.\\
9766     This-error-is-fatal.
9767 }
9768 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9769 {
9770     Unknown-cell.\\
9771     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9772     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9773     can't-be-executed-because~a~cell~doesn't-exist.\\
9774     This-command~\token_to_str:N \line\ will-be-ignored.
9775 }
9776 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9777 {

```

```

9778 Duplicate-name.\
9779 The-name-'\#1'-is-already-used-for-a-'\token_to_str:N \SubMatrix\
9780 in-this-'\@@_full_name_env:.\
9781 This-key-will-be-ignored.\
9782 \bool_if:NF \g_@@_messages_for_Overleaf_bool
9783 { For-a-list-of-the-names-already-used,-type-H<return>. }
9784 }
9785 {
9786 The-names-already-defined-in-this-'\@@_full_name_env:\ are:~
9787 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9788 }
9789 \@@_msg_new:nn { r-or-l-with-preamble }
9790 {
9791 Erroneous-use.\
9792 You-can't-use-the-key-'\l_keys_key_str'~in-your-'\@@_full_name_env:~
9793 You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
9794 your-'\@@_full_name_env:.\
9795 This-key-will-be-ignored.
9796 }
9797 \@@_msg_new:nn { Hdotsfor-in-col-0 }
9798 {
9799 Erroneous-use.\
9800 You-can't-use-'\token_to_str:N \Hdotsfor\ in-an-exterior-column-of~
9801 the-array.~This-error-is-fatal.
9802 }
9803 \@@_msg_new:nn { bad-corner }
9804 {
9805 Bad-corner.\
9806 #1-is-an-incorrect-specification-for-a-corner~(in-the-key~
9807 'corners').~The-available-values-are:~NW,~SW,~NE~and~SE.\
9808 This-specification-of-corner-will-be-ignored.
9809 }
9810 \@@_msg_new:nn { bad-border }
9811 {
9812 Bad-border.\
9813 \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
9814 (in-the-key~'borders'~of-the-command-'\token_to_str:N \Block).~
9815 The-available-values-are:~left,~right,~top~and~bottom~(and-you-can~
9816 also-use-the-key~'tikz'
9817 \IfPackageLoadedF { tikz }
9818 {~if-you-load-the-LaTeX-package~'tikz'}).~
9819 This-specification-of-border-will-be-ignored.
9820 }
9821 \@@_msg_new:nn { TikzEveryCell-without-tikz }
9822 {
9823 TikZ-not-loaded.\
9824 You-can't-use-'\token_to_str:N \TikzEveryCell\
9825 because-you-have-not-loaded-tikz.~
9826 This-command-will-be-ignored.
9827 }
9828 \@@_msg_new:nn { tikz-key-without-tikz }
9829 {
9830 TikZ-not-loaded.\
9831 You-can't-use-the-key~'tikz'~for-the-command-'\token_to_str:N
9832 \Block'~because-you-have-not-loaded-tikz.~
9833 This-key-will-be-ignored.
9834 }
9835 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
9836 {
9837 Erroneous-use.\
9838 In-the-'\@@_full_name_env:,~you-must-use-the-key~

```

```

9839     'last-col'~without~value.\\
9840     However,~you~can~go~on~for~this~time~
9841     (the~value~'\l_keys_value_tl'~will~be~ignored).
9842 }
9843 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
9844 {
9845     Erroneous~use.\\
9846     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9847     'last-col'~without~value.\\
9848     However,~you~can~go~on~for~this~time~
9849     (the~value~'\l_keys_value_tl'~will~be~ignored).
9850 }
9851 \@@_msg_new:nn { Block~too~large~1 }
9852 {
9853     Block~too~large.\\
9854     You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
9855     too~small~for~that~block. \\
9856     This~block~and~maybe~others~will~be~ignored.
9857 }
9858 \@@_msg_new:nn { Block~too~large~2 }
9859 {
9860     Block~too~large.\\
9861     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9862     \g_@@_static_num_of_col_int\
9863     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9864     specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
9865     (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\\
9866     This~block~and~maybe~others~will~be~ignored.
9867 }
9868 \@@_msg_new:nn { unknown~column~type }
9869 {
9870     Bad~column~type.\\
9871     The~column~type~'#1'~in~your~\@@_full_name_env:\
9872     is~unknown. \\
9873     This~error~is~fatal.
9874 }
9875 \@@_msg_new:nn { unknown~column~type~S }
9876 {
9877     Bad~column~type.\\
9878     The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9879     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9880     load~that~package. \\
9881     This~error~is~fatal.
9882 }
9883 \@@_msg_new:nn { tabularnote~forbidden }
9884 {
9885     Forbidden~command.\\
9886     You~can't~use~the~command~\token_to_str:N\tabularnote\
9887     ~here.~This~command~is~available~only~in~
9888     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9889     the~argument~of~a~command~\token_to_str:N \caption\ included~
9890     in~an~environment~{table}. \\
9891     This~command~will~be~ignored.
9892 }
9893 \@@_msg_new:nn { borders~forbidden }
9894 {
9895     Forbidden~key.\\
9896     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9897     because~the~option~'rounded~corners'~
9898     is~in~force~with~a~non~zero~value.\\
9899     This~key~will~be~ignored.

```

```

9900 }
9901 \@@_msg_new:nn { bottomrule~without~booktabs }
9902 {
9903   booktabs~not~loaded.\\
9904   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9905   loaded~'booktabs'.\\
9906   This~key~will~be~ignored.
9907 }
9908 \@@_msg_new:nn { enumitem~not~loaded }
9909 {
9910   enumitem~not~loaded.\\
9911   You~can't~use~the~command~\token_to_str:N\tabularnote\
9912   ~because~you~haven't~loaded~'enumitem'.\\
9913   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9914   ignored~in~the~document.
9915 }
9916 \@@_msg_new:nn { tikz~without~tikz }
9917 {
9918   Tikz~not~loaded.\\
9919   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9920   loaded.~If~you~go~on,~that~key~will~be~ignored.
9921 }
9922 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9923 {
9924   Tikz~not~loaded.\\
9925   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9926   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9927   You~can~go~on~but~you~will~have~another~error~if~you~actually~
9928   use~that~custom~line.
9929 }
9930 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9931 {
9932   Tikz~not~loaded.\\
9933   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9934   command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9935   That~key~will~be~ignored.
9936 }
9937 \@@_msg_new:nn { without~color~inside }
9938 {
9939   If~order~to~use~\token_to_str:N\cellcolor,~\token_to_str:N\rowcolor,~
9940   \token_to_str:N\rowcolors\ or~\token_to_str:N\rowlistcolors\
9941   outside~\token_to_str:N\CodeBefore,~you~
9942   should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\
9943   You~can~go~on~but~you~may~need~more~compilations.
9944 }
9945 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9946 {
9947   Erroneous~use.\\
9948   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9949   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9950   The~key~'color'~will~be~discarded.
9951 }
9952 \@@_msg_new:nn { Wrong~last~row }
9953 {
9954   Wrong~number.\\
9955   You~have~used~'last~row=\int_use:N\l_@@_last_row_int'~but~your~
9956   \@@_full_name_env:\ seems~to~have~\int_use:N\c@iRow\ rows.~
9957   If~you~go~on,~the~value~of~\int_use:N\c@iRow\ will~be~used~for~
9958   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9959   without~value~(more~compilations~might~be~necessary).

```

```

9960 }
9961 \@@_msg_new:nn { Yet~in~env }
9962 {
9963   Nested~environments.\
9964   Environments-of~nicematrix~can't~be~nested.\
9965   This~error~is~fatal.
9966 }
9967 \@@_msg_new:nn { Outside~math~mode }
9968 {
9969   Outside~math~mode.\
9970   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9971   (and~not~in~\token_to_str:N \vcenter).\
9972   This~error~is~fatal.
9973 }
9974 \@@_msg_new:nn { One~letter~allowed }
9975 {
9976   Bad~name.\
9977   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
9978   It~will~be~ignored.
9979 }
9980 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9981 {
9982   Environment~{TabularNote}~forbidden.\
9983   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9984   but~*before*~the~\token_to_str:N \CodeAfter.\
9985   This~environment~{TabularNote}~will~be~ignored.
9986 }
9987 \@@_msg_new:nn { varwidth~not~loaded }
9988 {
9989   varwidth~not~loaded.\
9990   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9991   loaded.\
9992   Your~column~will~behave~like~'p'.
9993 }
9994 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9995 {
9996   Unknow~key.\
9997   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
9998   \c_@@_available_keys_str
9999 }
10000 {
10001   The~available~keys~are~(in~alphabetic~order):~
10002   color,~
10003   dotted,~
10004   multiplicity,~
10005   sep-color,~
10006   tikz,~and~total-width.
10007 }
10008
10009 \@@_msg_new:nnn { Unknown~key~for~Block }
10010 {
10011   Unknown~key.\
10012   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
10013   \Block.\ It~will~be~ignored. \
10014   \c_@@_available_keys_str
10015 }
10016 {
10017   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10018   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10019   opacity,~rounded-corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10020   and~vlines.

```

```

10021 }
10022 \@@_msg_new:nnn { Unknown~key~for~Brace }
10023 {
10024   Unknown~key.\\
10025   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
10026   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
10027   It~will~be~ignored. \\
10028   \c_@@_available_keys_str
10029 }
10030 {
10031   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10032   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10033   right~shorten)~and~yshift.
10034 }
10035 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10036 {
10037   Unknown~key.\\
10038   The~key~'\l_keys_key_str'~is~unknown.\\
10039   It~will~be~ignored. \\
10040   \c_@@_available_keys_str
10041 }
10042 {
10043   The~available~keys~are~(in~alphabetic~order):~
10044   delimiters/color,~
10045   rules~(with~the~subkeys~'color'~and~'width'),~
10046   sub~matrix~(several~subkeys)~
10047   and~xdots~(several~subkeys).~
10048   The~latter~is~for~the~command~\token_to_str:N \line.
10049 }
10050 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10051 {
10052   Unknown~key.\\
10053   The~key~'\l_keys_key_str'~is~unknown.\\
10054   It~will~be~ignored. \\
10055   \c_@@_available_keys_str
10056 }
10057 {
10058   The~available~keys~are~(in~alphabetic~order):~
10059   create~cell~nodes,~
10060   delimiters/color~and~
10061   sub~matrix~(several~subkeys).
10062 }
10063 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10064 {
10065   Unknown~key.\\
10066   The~key~'\l_keys_key_str'~is~unknown.\\
10067   That~key~will~be~ignored. \\
10068   \c_@@_available_keys_str
10069 }
10070 {
10071   The~available~keys~are~(in~alphabetic~order):~
10072   'delimiters/color',~
10073   'extra~height',~
10074   'hlines',~
10075   'hvlines',~
10076   'left~xshift',~
10077   'name',~
10078   'right~xshift',~
10079   'rules'~(with~the~subkeys~'color'~and~'width'),~
10080   'slim',~
10081   'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
10082   and~'right~xshift').\\

```



```

10083 }
10084 \@@_msg_new:nnn { Unknown~key~for~notes }
10085 {
10086   Unknown~key.\\
10087   The~key~'\l_keys_key_str'~is~unknown.\\
10088   That~key~will~be~ignored. \\
10089   \c_@@_available_keys_str
10090 }
10091 {
10092   The~available~keys~are~(in~alphabetic~order):~
10093   bottomrule,~
10094   code~after,~
10095   code~before,~
10096   detect~duplicates,~
10097   enumitem~keys,~
10098   enumitem~keys~para,~
10099   para,~
10100   label~in~list,~
10101   label~in~tabular~and~
10102   style.
10103 }
10104 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10105 {
10106   Unknown~key.\\
10107   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10108   \token_to_str:N \RowStyle. \\
10109   That~key~will~be~ignored. \\
10110   \c_@@_available_keys_str
10111 }
10112 {
10113   The~available~keys~are~(in~alphabetic~order):~
10114   'bold',~
10115   'cell-space-top-limit',~
10116   'cell-space-bottom-limit',~
10117   'cell-space-limits',~
10118   'color',~
10119   'nb-rows'~and~
10120   'rowcolor'.
10121 }
10122 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10123 {
10124   Unknown~key.\\
10125   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10126   \token_to_str:N \NiceMatrixOptions. \\
10127   That~key~will~be~ignored. \\
10128   \c_@@_available_keys_str
10129 }
10130 {
10131   The~available~keys~are~(in~alphabetic~order):~
10132   &-in-blocks,~
10133   allow~duplicate~names,~
10134   ampersand~in~blocks,~
10135   caption~above,~
10136   cell-space-bottom-limit,~
10137   cell-space-limits,~
10138   cell-space-top-limit,~
10139   code~for~first~col,~
10140   code~for~first~row,~
10141   code~for~last~col,~
10142   code~for~last~row,~
10143   corners,~
10144   custom~key,~
10145   create~extra~nodes,~

```

```

10146     create-medium-nodes,~
10147     create-large-nodes,~
10148     custom-line,~
10149     delimiters~(several~subkeys),~
10150     end-of-row,~
10151     first-col,~
10152     first-row,~
10153     hlines,~
10154     hvlines,~
10155     hvlines-except-borders,~
10156     last-col,~
10157     last-row,~
10158     left-margin,~
10159     light-syntax,~
10160     light-syntax-expanded,~
10161     matrix/columns-type,~
10162     no-cell-nodes,~
10163     notes~(several~subkeys),~
10164     nullify-dots,~
10165     pgf-node-code,~
10166     renew-dots,~
10167     renew-matrix,~
10168     respect-arraystretch,~
10169     rounded-corners,~
10170     right-margin,~
10171     rules~(with~the~subkeys~'color'~and~'width'),~
10172     small,~
10173     sub-matrix~(several~subkeys),~
10174     vlins,~
10175     xdots~(several~subkeys).
10176 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10177 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10178 {
10179     Unknown~key.\\
10180     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10181     \{NiceArray\}. \\
10182     That~key~will~be~ignored. \\
10183     \c_@@_available_keys_str
10184 }
10185 {
10186     The~available~keys~are~(in~alphabetic~order):~
10187     &~in~blocks,~
10188     ampersand~in~blocks,~
10189     b,~
10190     baseline,~
10191     c,~
10192     cell-space-bottom-limit,~
10193     cell-space-limits,~
10194     cell-space-top-limit,~
10195     code-after,~
10196     code-for-first-col,~
10197     code-for-first-row,~
10198     code-for-last-col,~
10199     code-for-last-row,~
10200     color-inside,~
10201     columns-width,~
10202     corners,~
10203     create-extra-nodes,~
10204     create-medium-nodes,~
10205     create-large-nodes,~
10206     extra-left-margin,~

```

```

10207     extra-right-margin,~
10208     first-col,~
10209     first-row,~
10210     hlines,~
10211     hvlines,~
10212     hvlines-except-borders,~
10213     last-col,~
10214     last-row,~
10215     left-margin,~
10216     light-syntax,~
10217     light-syntax-expanded,~
10218     name,~
10219     no-cell-nodes,~
10220     nullify-dots,~
10221     pgf-node-code,~
10222     renew-dots,~
10223     respect-arraystretch,~
10224     right-margin,~
10225     rounded-corners,~
10226     rules~(with~the~subkeys~'color'~and~'width'),~
10227     small,~
10228     t,~
10229     vlines,~
10230     xdots/color,~
10231     xdots/shorten-start,~
10232     xdots/shorten-end,~
10233     xdots/shorten-and~
10234     xdots/line-style.
10235 }

```

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`).

```

10236 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10237 {
10238   Unknown~key.\\
10239   The~key~'\l_keys_key_str'~is~unknown~for~the~
10240   \@@_full_name_env:. \\
10241   That~key~will~be~ignored. \\
10242   \c_@@_available_keys_str
10243 }
10244 {
10245   The~available~keys~are~(in~alphabetic~order):~
10246   &~in~blocks,~
10247   ampersand~in~blocks,~
10248   b,~
10249   baseline,~
10250   c,~
10251   cell-space-bottom-limit,~
10252   cell-space-limits,~
10253   cell-space-top-limit,~
10254   code-after,~
10255   code-for-first-col,~
10256   code-for-first-row,~
10257   code-for-last-col,~
10258   code-for-last-row,~
10259   color-inside,~
10260   columns-type,~
10261   columns-width,~
10262   corners,~
10263   create-extra-nodes,~
10264   create-medium-nodes,~
10265   create-large-nodes,~
10266   extra-left-margin,~
10267   extra-right-margin,~

```

```

10268 first-col,~
10269 first-row,~
10270 hlines,~
10271 hvlines,~
10272 hvlines-except-borders,~
10273 l,~
10274 last-col,~
10275 last-row,~
10276 left-margin,~
10277 light-syntax,~
10278 light-syntax-expanded,~
10279 name,~
10280 no-cell-nodes,~
10281 nullify-dots,~
10282 pgf-node-code,~
10283 r,~
10284 renew-dots,~
10285 respect-arraystretch,~
10286 right-margin,~
10287 rounded-corners,~
10288 rules~(with~the~subkeys~'color'~and~'width'),~
10289 small,~
10290 t,~
10291 vlines,~
10292 xdots/color,~
10293 xdots/shorten-start,~
10294 xdots/shorten-end,~
10295 xdots/shorten-and~
10296 xdots/line-style.
10297 }
10298 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10299 {
10300   Unknown~key.\\
10301   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10302   \{NiceTabular\}. \\
10303   That~key~will~be~ignored. \\
10304   \c_@@_available_keys_str
10305 }
10306 {
10307   The~available~keys~are~(in~alphabetic~order):~
10308   &in-blocks,~
10309   ampersand-in-blocks,~
10310   b,~
10311   baseline,~
10312   c,~
10313   caption,~
10314   cell-space-bottom-limit,~
10315   cell-space-limits,~
10316   cell-space-top-limit,~
10317   code-after,~
10318   code-for-first-col,~
10319   code-for-first-row,~
10320   code-for-last-col,~
10321   code-for-last-row,~
10322   color-inside,~
10323   columns-width,~
10324   corners,~
10325   custom-line,~
10326   create-extra-nodes,~
10327   create-medium-nodes,~
10328   create-large-nodes,~
10329   extra-left-margin,~
10330   extra-right-margin,~

```

```

10331 first-col,~
10332 first-row,~
10333 hlines,~
10334 hvlines,~
10335 hvlines-except-borders,~
10336 label,~
10337 last-col,~
10338 last-row,~
10339 left-margin,~
10340 light-syntax,~
10341 light-syntax-expanded,~
10342 name,~
10343 no-cell-nodes,~
10344 notes~(several~subkeys),~
10345 nullify-dots,~
10346 pgf-node-code,~
10347 renew-dots,~
10348 respect-arraystretch,~
10349 right-margin,~
10350 rounded-corners,~
10351 rules~(with~the~subkeys~'color'~and~'width'),~
10352 short-caption,~
10353 t,~
10354 tabularnote,~
10355 vlines,~
10356 xdots/color,~
10357 xdots/shorten-start,~
10358 xdots/shorten-end,~
10359 xdots/shorten-and~
10360 xdots/line-style.
10361 }

10362 \@@_msg_new:nnn { Duplicate~name }
10363 {
10364 Duplicate~name.\\
10365 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10366 the~same~environment~name~twice.~You~can~go~on,~but,~
10367 maybe,~you~will~have~incorrect~results~especially~
10368 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10369 message~again,~use~the~key~'allow-duplicate-names'~in~
10370 '\token_to_str:N \NiceMatrixOptions'.\\
10371 \bool_if:NF \g_@@_messages_for_Overleaf_bool
10372 { For~a~list~of~the~names~already~used,~type~H~<return>. }
10373 }
10374 {
10375 The~names~already~defined~in~this~document~are:~
10376 \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10377 }

10378 \@@_msg_new:nn { Option~auto~for~columns-width }
10379 {
10380 Erroneous~use.\\
10381 You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10382 That~key~will~be~ignored.
10383 }

10384 \@@_msg_new:nn { NiceTabularX~without~X }
10385 {
10386 NiceTabularX~without~X.\\
10387 You~should~not~use~{NiceTabularX}~without~X~columns.\\
10388 However,~you~can~go~on.
10389 }

10390 \@@_msg_new:nn { Preamble~forgotten }
10391 {
10392 Preamble~forgotten.\\

```

```
10393     You~have~probably~forgotten~the~preamble~of~your~
10394     \@@_full_name_env:. \
10395     This~error~is~fatal.
10396 }
```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	3
3	Collecting options	4
4	Technical definitions	5
5	Parameters	10
6	The command <code>\tabularnote</code>	20
7	Command for creation of rectangle nodes	25
8	The options	26
9	Important code used by <code>{NiceArrayWithDelims}</code>	37
10	The <code>\CodeBefore</code>	50
11	The environment <code>{NiceArrayWithDelims}</code>	54
12	We construct the preamble of the array	59
13	The redefinition of <code>\multicolumn</code>	75
14	The environment <code>{NiceMatrix}</code> and its variants	93
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	94
16	After the construction of the array	95
17	We draw the dotted lines	102
18	The actual instructions for drawing the dotted lines with Tikz	115
19	User commands available in the new environments	121
20	The command <code>\line</code> accessible in code-after	127
21	The command <code>\RowStyle</code>	128
22	Colors of cells, rows and columns	131
23	The vertical and horizontal rules	143
24	The empty corners	158
25	The environment <code>{NiceMatrixBlock}</code>	161
26	The extra nodes	162
27	The blocks	166
28	How to draw the dotted lines transparently	190
29	Automatic arrays	190
30	The redefinition of the command <code>\dotfill</code>	192

31	The command <code>\diagbox</code>	192
32	The keyword <code>\CodeAfter</code>	194
33	The delimiters in the preamble	194
34	The command <code>\SubMatrix</code>	196
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	204
36	The command <code>TikzEveryCell</code>	207
37	The command <code>\ShowCellNames</code>	208
38	We process the options at package loading	211
39	About the package underscore	213
40	Error messages of the package	213