

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

December 4, 2023

Abstract

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

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

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

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

We load some packages.

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

*This document corresponds to the version 6.26a of `nicematrix`, at the date of 2023/12/04.

```

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

```

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

```

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

```

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

```

25 \cs_new_protected:Npn \@@_error_or_warning:n
26 { \bool_if:NTF \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”.

```

27 \bool_new:N \g_@@_messages_for_Overleaf_bool
28 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
29 {
30   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
31   || \str_if_eq_p:on \c_sys_jobname_str { output } % for Overleaf
32 }

```

```

33 \cs_new_protected:Npn \@@_msg_redirect_name:nn
34 { \msg_redirect_name:nnn { nicematrix } }
35 \cs_new_protected:Npn \@@_gredirect_none:n #1
36 {
37   \group_begin:
38   \globaldefs = 1
39   \@@_msg_redirect_name:nn { #1 } { none }
40   \group_end:
41 }
42 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
43 {
44   \@@_error:n { #1 }
45   \@@_gredirect_none:n { #1 }
46 }
47 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
48 {
49   \@@_warning:n { #1 }
50   \@@_gredirect_none:n { #1 }
51 }

```

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.

```

52 \@@_msg_new:nn { Internal~error }
53 {
54   Potential~problem~when~using~nicematrix.\\
55   The~package~nicematrix~have~detected~a~modification~of~the~
56   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
57   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
58   this~message~again,~load~nicematrix~with:~\token_to_str:N
59   \usepackage[no-test-for-array]{nicematrix}.
60 }

61 \@@_msg_new:nn { mdwtab~loaded }
62 {
63   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
64   This~error~is~fatal.
65 }

66 \cs_new_protected:Npn \@@_security_test:n #1
67 {
68   \peek_meaning:NTF \ignorespaces
69   { \@@_security_test_i:w }
70   { \@@_error:n { Internal~error } }
71   #1
72 }

73 \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
74 {
75   \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
76   #1
77 }

```

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

```

78 \hook_gput_code:nnn { begindocument / after } { . }
79 {
80   \IfPackageLoadedTF { mdwtab }
81   { \@@_fatal:n { mdwtab~loaded } }
82   {
83     \bool_if:NF \g_@@_no_test_for_array_bool
84     {
85       \group_begin:
86       \hbox_set:Nn \l_tmpa_box
87       {
88         \begin { tabular } { c > { \@@_security_test:n } c c }
89         text & & text
90         \end { tabular }
91       }
92       \group_end:
93     }
94   }
95 }

```

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`).

```

96 \cs_new_protected:Npn \@@_collect_options:n #1
97 {
98   \peek_meaning:NTF [
99     { \@@_collect_options:nw { #1 } }
100     { #1 { } }
101   }

```

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

```

102 \NewDocumentCommand \@@_collect_options:nw { m r[] }
103 { \@@_collect_options:nn { #1 } { #2 } }
104
105 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
106 {
107   \peek_meaning:NTF [
108     { \@@_collect_options:nnw { #1 } { #2 } }
109     { #1 { #2 } }
110   }
111
112 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
113 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

4 Technical definitions

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

```

114 \tl_const:Nn \c_@@_b_tl { b }
115 \tl_const:Nn \c_@@_c_tl { c }
116 \tl_const:Nn \c_@@_l_tl { l }
117 \tl_const:Nn \c_@@_r_tl { r }
118 \tl_const:Nn \c_@@_all_tl { all }
119 \tl_const:Nn \c_@@_dot_tl { . }
120 \tl_const:Nn \c_@@_default_tl { default }
121 \tl_const:Nn \c_@@_star_tl { * }
122 \str_const:Nn \c_@@_r_str { r }
123 \str_const:Nn \c_@@_c_str { c }
124 \str_const:Nn \c_@@_l_str { l }
125 \str_const:Nn \c_@@_R_str { R }
126 \str_const:Nn \c_@@_C_str { C }
127 \str_const:Nn \c_@@_L_str { L }
128 \str_const:Nn \c_@@_j_str { j }
129 \str_const:Nn \c_@@_si_str { si }

```

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).

```

130 \tl_new:N \l_@@_argspec_tl
131 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
132 \cs_generate_variant:Nn \str_lowercase:n { V }

133 \hook_gput_code:nnn { begindocument } { . }
134 {
135   \IfPackageLoadedTF { tikz }
136   {

```

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

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

```

137   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
138   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
139 }
140 {
141   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
142   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
143 }
144 }

```

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 May 2023, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

145 \IfClassLoadedTF { revtex4-1 }
146 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
147 {
148   \IfClassLoadedTF { revtex4-2 }
149   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
150   {

```

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

```

151   \cs_if_exist:NT \rvtx@ifformat@geq
152   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
153   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
154 }
155 }

```

```

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

```

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.

```

157 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
158 {
159   \iow_now:Nn \@mainaux
160   {
161     \ExplSyntaxOn
162     \cs_if_free:NT \pgfsyspdfmark
163     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
164     \ExplSyntaxOff

```

```

165     }
166     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
167 }

```

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.

```

168 \ProvideDocumentCommand \iddots { }
169 {
170     \mathinner
171     {
172         \tex_mkern:D 1 mu
173         \box_move_up:nn { 1 pt } { \hbox { . } }
174         \tex_mkern:D 2 mu
175         \box_move_up:nn { 4 pt } { \hbox { . } }
176         \tex_mkern:D 2 mu
177         \box_move_up:nn { 7 pt }
178         { \vbox:n { \kern 7 pt \hbox { . } } }
179         \tex_mkern:D 1 mu
180     }
181 }

```

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.

```

182 \hook_gput_code:nnn { begindocument } { . }
183 {
184     \IfPackageLoadedTF { booktabs }
185     { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
186     { }
187 }
188 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
189 {
190     \cs_set_eq:NN \@@_old_pgfutil@check@rerun \pgfutil@check@rerun

```

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

```

191     \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
192     {
193         \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
194         { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
195     }
196 }

```

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

```

197 \hook_gput_code:nnn { begindocument } { . }
198 {
199     \IfPackageLoadedTF { colortbl }
200     { }
201     {

```

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.

```

202         \cs_set_protected:Npn \CT@arc@ { }
203         \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
204         \cs_set:Npn \CT@arc@ #1 #2
205         {
206             \dim_compare:nNt \baselineskip = \c_zero_dim \noalign
207             { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
208         }

```

Idem for `\CT@drs@`.

```

209     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
210     \cs_set:Npn \CT@drs #1 #2
211     {
212         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
213         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
214     }
215     \cs_set:Npn \hline
216     {
217         \noalign { \ifnum 0 = ` } \fi
218         \cs_set_eq:NN \hskip \vskip
219         \cs_set_eq:NN \vrule \hrule
220         \cs_set_eq:NN \@width \@height
221         { \CT@arc@ \vline }
222         \futurelet \reserved@a
223         \@xhline
224     }
225 }
226 }
```

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.

```

227 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
228 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
229 {
230     \int_if_zero:nT \l_@@_first_col_int { \omit & }
231     \int_compare:nNnT { #1 } > \c_one_int
232     { \multispan { \int_eval:n { #1 - 1 } } & }
233     \multispan { \int_eval:n { #2 - #1 + 1 } }
234     {
235         \CT@arc@
236         \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

237     \skip_horizontal:N \c_zero_dim
238 }
```

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`.

```

239     \everycr { }
240     \cr
241     \noalign { \skip_vertical:N -\arrayrulewidth }
242 }
```

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.

```

243 \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`.

```

244 { \@@_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*.

```

245 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
246 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
247 {
```

¹See question 99041 on TeX StackExchange.

```

248 \tl_if_empty:nTF { #3 }
249 { \@@_cline_iii:w #1|#2-#2 \q_stop }
250 { \@@_cline_ii:w #1|#2-#3 \q_stop }
251 }
252 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
253 { \@@_cline_iii:w #1|#2-#3 \q_stop }
254 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
255 {

```

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).

```

256 \int_compare:nNt { #1 } < { #2 }
257 { \multispan { \int_eval:n { #2 - #1 } } & }
258 \multispan { \int_eval:n { #3 - #2 + 1 } }
259 {
260 \CT@arc@
261 \leaders \hrule \@height \arrayrulewidth \hfill
262 \skip_horizontal:N \c_zero_dim
263 }

```

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

```

264 \peek_meaning_remove_ignore_spaces:NTF \cline
265 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
266 { \everycr { } \cr }
267 }
268 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

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

```

269 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

```

```

270 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
271 {
272 \tl_if_blank:nF { #1 }
273 {
274 \tl_if_head_eq_meaning:nNTF { #1 } [
275 { \cs_set:Npn \CT@arc@ { \color #1 } }
276 { \cs_set:Npn \CT@arc@ { \color { #1 } } }
277 ]
278 }
279 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }

```

```

280 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
281 {
282 \tl_if_head_eq_meaning:nNTF { #1 } [
283 { \cs_set:Npn \CT@drsc@ { \color #1 } }
284 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
285 ]
286 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }

```

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

```

287 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
288 {
289 \tl_if_head_eq_meaning:nNTF { #2 } [
290 { #1 #2 }
291 { #1 { #2 } }
292 ]
293 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```


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

```

294 \cs_new_protected:Npn \@@_color:n #1
295   { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
296 \cs_generate_variant:Nn \@@_color:n { o }

297 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor

298 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
299   {
300     \tl_set_rescan:Nno
301       #1
302     {
303       \char_set_catcode_other:N >
304       \char_set_catcode_other:N <
305     }
306     #1
307   }

```

Since we will do ourself the expansion of the preamble of the array, we will modify `\@mkpream` of array in order to skip the operation of expansion done by `\@mkpream`.

```

308 \cs_set_eq:NN \@@_old_mkpream: \@mkpream
309 \cs_set_protected:Npn \@@_mkpream: #1
310   {

```

The command `\@@_mkpream_colortbl:` will be empty when `colortbl` is not loaded.

```

311   \@mkpream_colortbl:
312   \gdef\@preamble{} \@lastchclass 4 \@firstampttrue
313   \let\@sharp\relax
314   \def\@startpbox##1{\unexpanded\expandafter{\expandafter
315     \@startpbox\expandafter{##1}}}\let\@endpbox\relax
316   \let\do@row@strut\relax
317   \let\ar@align@mcell\relax
318   \@temptokena{#1} % \@tempswatrue
319   % \@whiles\if@tempswa\fi{\@tempswafalse\the\NC@list}%
320   \count@m@ne
321   \let\the@toks\relax
322   \prepnext@tok

```

We have slightly modified the code of the original version of `\@mkpream` in order to have something compatible with `\ExplSyntaxOn`.

```

323   \exp_args:NV \tl_map_variable:NNn \@temptokena \@nextchar
324   {\@testpach
325     \ifcase \@chclass \@classz \or \@classi \or \@classii
326       \or \save@decl \or \or \@classv \or \@classvi
327       \or \@classvii \or \@classviii
328       \or \@classx
329       \or \@classx \fi
330     \@lastchclass\@chclass}%
331   \ifcase\@lastchclass
332     \@acol \or
333     \or
334     \@acol \or
335     \@preamerr \thr@@ \or
336     \@preamerr \tw@ \@addtopreamble\@sharp \or
337     \or
338     \else \@preamerr \@ne \fi
339     \def\the@toks{\the\toks}

```

After an utilisation of the modified version of `\@mkpream`, we come back to the original version because there may be occurrences of the classical `{array}` in the cells of our array (of `nicematrix`).

```

340   \cs_gset_eq:NN \@mkpream \@@_old_mkpream:
341   }

```

The classes of REVTeX do their own redefinition of `\array` and that's why the previous mechanism is not compatible with REVTeX. However, it would probably be possible to do something similar for REVTeX...

```

342 \hook_gput_code:nnn { begindocument } { . . }
343 {
344   \bool_if:NTF \c_@@_revtex_bool
345   { \cs_set_eq:NN \@@_redefine_mkpream: \prg_do_nothing: }
346   {
347     \IfPackageLoadedTF { arydshln }
348     { \cs_set_eq:NN \@@_redefine_mkpream: \prg_do_nothing: }
349     {
350       \cs_new_protected:Npn \@@_redefine_mkpream:
351       { \cs_set_eq:NN \@mkpream \@@_mkpream: }
352     }
353   }
354 }

355 \cs_new_protected:Npn \@@_mkpream_colortbl: { }
356 \hook_gput_code:nnn { begindocument } { . . }
357 {
358   \IfPackageLoadedTF { colortbl }
359   {
360     \cs_set_protected:Npn \@@_mkpream_colortbl:
361     {

```

The following lines are a patch added to `\@mkpream` by `colortbl` (by storing the version of `\@mkpream` provided by `array` in `\@mkpreamarray`). Since you do a redefinition of `\@mkpream`, you have to add the following lines in our redefinition when `colortbl` is loaded.

```

362       \cs_set_eq:NN \CT@setup \relax
363       \cs_set_eq:NN \CT@color \relax
364       \cs_set_eq:NN \CT@do@color \relax
365       \cs_set_eq:NN \color \relax
366       \cs_set_eq:NN \CT@column@color \relax
367       \cs_set_eq:NN \CT@row@color \relax
368       \cs_set_eq:NN \CT@cell@color \relax
369     }
370   }
371   { }
372 }

```

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.

```

373 \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).

```

374 \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.

```

375 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
376 { \int_use:N \g_@@_env_int }

```

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

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

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

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

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

```
380 \bool_new:N \g_@@_delims_bool
381 \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.

```
382 \bool_new:N \l_@@_preamble_bool
383 \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.

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

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

```
385 \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.

```
386 \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).

```
387 \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.).

```
388 \dim_new:N \l_@@_col_width_dim
389 \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.

```
390 \int_new:N \g_@@_row_total_int
391 \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).

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

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

```
393 \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.

```
394 \tl_new:N \l_@@_hpos_cell_tl
395 \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`.

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

Idem for the mono-row blocks.

```
397 \dim_new:N \g_@@_blocks_ht_dim
398 \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}`).

```
399 \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`.

```
400 \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.

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

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

```
402 \bool_new:N \l_@@_notes_detect_duplicates_bool
403 \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.

```
404 \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).

```
405 \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).

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

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

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

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

```
408 \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.

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

```
410 \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 }`).

```
411 \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.

```
412 \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 the following sequence that will contain informations about the size of the array.

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

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

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

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

```
416 \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`).

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

For `\multicolumn`.

```
418 \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`.

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

```
420 \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 `:`.

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

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

```
423 \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`).

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

```
425 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
426 {
```

```
427   \if_mode_math: \else:
```

```
428     \@@_fatal:n { Outside-math-mode }
```

```
429   \fi:
```

```
430 }
```

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.

```
431 \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”.

```
432 \colorlet { nicematrix-last-col } { . }
433 \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`).

```
434 \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*.

```
435 \tl_new:N \g_@@_com_or_env_str
436 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

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

```
437 \cs_new:Npn \@@_full_name_env:
438 {
439   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
440   { command \space \c_backslash_str \g_@@_name_env_str }
441   { environment \space \{ \g_@@_name_env_str \} }
442 }
```

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

```
443 \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.

```
444 \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.

```
445 \tl_new:N \g_@@_pre_code_before_tl
446 \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.

```
447 \tl_new:N \g_@@_pre_code_after_tl
448 \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.

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

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

```
450 \int_new:N \l_@@_old_iRow_int
451 \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).

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

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

```
453 \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.

```
454 \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.

```
455 \bool_new:N \l_@@_X_columns_aux_bool
456 \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.

```
457 \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).

```
458 \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).

```
459 \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`).

```
460 \tl_new:N \l_@@_code_before_tl
461 \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).

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

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

```
463 \dim_new:N \l_@@_x_initial_dim
464 \dim_new:N \l_@@_y_initial_dim
465 \dim_new:N \l_@@_x_final_dim
466 \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.

```
467 \dim_zero_new:N \l_@@_tmpc_dim
468 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
469 \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”.

```
470 \dim_new:N \g_@@_width_last_col_dim
471 \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.

```
472 \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!).

```
473 \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}`.

```
474 \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.

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

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

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

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

```
477 \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.

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

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

```
479 \seq_new:N \g_@@_multicolumn_cells_seq
480 \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).

```

481 \int_new:N \l_@@_row_min_int
482 \int_new:N \l_@@_row_max_int
483 \int_new:N \l_@@_col_min_int
484 \int_new:N \l_@@_col_max_int

```

The following counters will be used when drawing the rules.

```

485 \int_new:N \l_@@_start_int
486 \int_set_eq:NN \l_@@_start_int \c_one_int
487 \int_new:N \l_@@_end_int
488 \int_new:N \l_@@_local_start_int
489 \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.

```

490 \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).

```

491 \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`.

```

492 \tl_new:N \l_@@_fill_tl
493 \tl_new:N \l_@@_opacity_tl
494 \tl_new:N \l_@@_draw_tl
495 \seq_new:N \l_@@_tikz_seq
496 \clist_new:N \l_@@_borders_clist
497 \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.

```

498 \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`.

```

499 \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.

```

500 \dim_new:N \l_@@_offset_dim

```

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

```

501 \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`).

```

502 \str_new:N \l_@@_hpos_block_str
503 \str_set:Nn \l_@@_hpos_block_str { c }
504 \bool_new:N \l_@@_hpos_of_block_cap_bool

```

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

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

For the vertical position, the possible values are c, t and b.

```
506 \str_new:N \l_@@_vpos_block_str
507 \str_set:Nn \l_@@_vpos_block_str { c }
```

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

```
508 \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).

```
509 \bool_new:N \l_@@_vlines_block_bool
510 \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.

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

512 \dim_new:N \l_@@_submatrix_extra_height_dim
513 \dim_new:N \l_@@_submatrix_left_xshift_dim
514 \dim_new:N \l_@@_submatrix_right_xshift_dim
515 \clist_new:N \l_@@_hlines_clist
516 \clist_new:N \l_@@_vlines_clist
517 \clist_new:N \l_@@_submatrix_hlines_clist
518 \clist_new:N \l_@@_submatrix_vlines_clist
```

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

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

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

```
520 \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).

```
521 \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.

```
522 \int_new:N \l_@@_first_row_int
523 \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.

```
524 \int_new:N \l_@@_first_col_int
525 \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).

```
526 \int_new:N \l_@@_last_row_int
527 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
529 \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`.

```
530 \int_new:N \l_@@_last_col_int
531 \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.

```
532 \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`).

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

Some utilities

```
534 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
535 {
536   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
537   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
538 }
```

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.

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.

```

539 \cs_new_protected:Npn \@@_expand_clist:N #1
540 {
541   \clist_if_in:NVF #1 \c_@@_all_tl
542   {
543     \clist_clear:N \l_tmpa_clist
544     \clist_map_inline:Nn #1
545     {
546       \tl_if_in:nnTF { ##1 } { - }
547       { \@@_cut_on_hyphen:w ##1 \q_stop }
548       {
549         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
550         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
551       }
552       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
553       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
554     }
555     \tl_set_eq:NN #1 \l_tmpa_clist
556   }
557 }
```

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.

```

558 \hook_gput_code:nnn { begindocument } { . }
559 {
560   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
561   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
562   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
563 }
```

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 composant of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabulernote}`. The first composante is the optional argument (between square brackets) of the command `\tabulernote` (if the optional argument is not used, the value will be the special marker `\c_novalue_tl`).
- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the composantes of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

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

```

564 \newcounter { tabulernote }
565 \seq_new:N \g_@@_notes_seq
566 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

567 \tl_new:N \g_@@_tabulernote_tl

```

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

```

568 \seq_new:N \l_@@_notes_labels_seq
569 \newcounter{nicematrix_draft}
570 \cs_new_protected:Npn \@@_notes_format:n #1
571 {
572   \setcounter { nicematrix_draft } { #1 }
573   \@@_notes_style:n { nicematrix_draft }
574 }

```

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

```

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

```

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

```

576 \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`.

```

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

```

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

```

578 \cs_set:Npn \thetabulernote { { \@@_notes_style:n { tabulernote } } }

```

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

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

```

579 \hook_gput_code:nnn { begindocument } { . }
580 {
581   \IfPackageLoadedTF { enumitem }
582   {

```

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.

```

583     \newlist { tabularnotes } { enumerate } { 1 }
584     \setlist [ tabularnotes ]
585     {
586       topsep = 0pt ,
587       noitemsep ,
588       leftmargin = * ,
589       align = left ,
590       labelsep = 0pt ,
591       label =
592         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
593     }
594     \newlist { tabularnotes* } { enumerate* } { 1 }
595     \setlist [ tabularnotes* ]
596     {
597       afterlabel = \nobreak ,
598       itemjoin = \quad ,
599       label =
600         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
601     }

```

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`.

```

602     \NewDocumentCommand \tabularnote { o m }
603     {
604       \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } \l_@@_in_env_bool
605       {
606         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
607         { \@@_error:n { tabularnote~forbidden } }
608         {
609           \bool_if:NTF \l_@@_in_caption_bool
610             \@@_tabularnote_caption:nn
611             \@@_tabularnote:nn
612             { #1 } { #2 }
613         }
614       }
615     }
616   }
617 {
618   \NewDocumentCommand \tabularnote { o m }
619   {
620     \@@_error_or_warning:n { enumitem~not~loaded }
621     \@@_gredirect_none:n { enumitem~not~loaded }
622   }
623 }
624 }

```

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

argument of `\tabularnote`.

```
625 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
626 {
```

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).

```
627 \int_zero:N \l_tmpa_int
628 \bool_if:NT \l_@@_notes_detect_duplicates_bool
629 {
```

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

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

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

```
630 \seq_map_indexed_inline:Nn \g_@@_notes_seq
631 {
632   \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
633   {
634     \int_set:Nn \l_tmpa_int { ##1 }
635     \seq_map_break:
636   }
637 }
638 \int_if_zero:nF \l_tmpa_int
639 { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
640 }
641 \int_if_zero:nT \l_tmpa_int
642 {
643   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
644   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
645 }
646 \seq_put_right:Nx \l_@@_notes_labels_seq
647 {
648   \tl_if_novalue:nTF { #1 }
649   {
650     \@@_notes_format:n
651     {
652       \int_eval:n
653       {
654         \int_if_zero:nTF \l_tmpa_int
655         \c@tabularnote
656         \l_tmpa_int
657       }
658     }
659   }
660   { #1 }
661 }
662 \peek_meaning:NF \tabularnote
663 {
```

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

```
664 \hbox_set:Nn \l_tmpa_box
665 {
```

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

```
666 \@@_notes_label_in_tabular:n
```

```

667         {
668         \seq_use:Nnnn
669         \l_@@_notes_labels_seq { , } { , } { , }
670         }
671     }

```

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

```

672     \int_gdecr:N \c@tabularnote
673     \int_set_eq:NN \l_tmpa_int \c@tabularnote
674     \refstepcounter { tabularnote }
675     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
676     { \int_gincr:N \c@tabularnote }
677     \seq_clear:N \l_@@_notes_labels_seq
678     \bool_lazy_or:nnTF
679     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
680     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
681     {
682         \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).

```

683         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
684     }
685     { \box_use:N \l_tmpa_box }
686 }
687 }

```

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`.

```

688 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
689 {
690     \bool_if:NTF \g_@@_caption_finished_bool
691     {
692         \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
693         { \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`!

```

694         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
695         { \@@_error:n { Identical~notes~in~caption } }
696     }
697 }

```

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

```

698     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
699     {

```

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.

```

700         \bool_gset_true:N \g_@@_caption_finished_bool
701         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
702         \int_gzero:N \c@tabularnote
703     }
704     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
705 }

```


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!

```

706 \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
707 \seq_put_right:Nx \l_@@_notes_labels_seq
708 {
709   \tl_if_novalue:nTF { #1 }
710     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
711     { #1 }
712 }
713 \peek_meaning:NF \tabularnote
714 {
715   \@@_notes_label_in_tabular:n
716     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
717   \seq_clear:N \l_@@_notes_labels_seq
718 }
719 }

720 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
721 { \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.

```

722 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
723 {
724   \begin { pgfscope }
725   \pgfset
726   {
727     inner~sep = \c_zero_dim ,
728     minimum~size = \c_zero_dim
729   }
730   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
731   \pgfnode
732   { rectangle }
733   { center }
734   {
735     \vbox_to_ht:nn
736       { \dim_abs:n { #5 - #3 } }
737       {
738         \vfill
739         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
740       }
741   }
742   { #1 }
743   { }
744   \end { pgfscope }
745 }

```

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.

```

746 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
747 {
748   \begin { pgfscope }
749   \pgfset
750   {
751     inner~sep = \c_zero_dim ,
752     minimum~size = \c_zero_dim

```

```

753     }
754     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
755     \pgfpointdiff { #3 } { #2 }
756     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
757     \pgfnode
758     { rectangle }
759     { center }
760     {
761         \vbox_to_ht:nn
762         { \dim_abs:n \l_tmpb_dim }
763         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
764     }
765     { #1 }
766     { }
767     \end { pgfscope }
768 }

```

8 The options

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

```

769 \tl_new:N \l_@@_caption_tl
770 \tl_new:N \l_@@_short_caption_tl
771 \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).

```

772 \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.

```

773 \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`.

```

774 \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`).

```

775 \dim_new:N \l_@@_cell_space_top_limit_dim
776 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

777 \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.

```

778 \dim_new:N \l_@@_xdots_inter_dim
779 \hook_gput_code:nnn { begindocument } { . }
780 { \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).

```

781 \dim_new:N \l_@@_xdots_shorten_start_dim
782 \dim_new:N \l_@@_xdots_shorten_end_dim
783 \hook_gput_code:nnn { begindocument } { . }
784 {
785   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
786   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
787 }
```

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.

```

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

```

791 \tl_new:N \l_@@_xdots_line_style_tl
792 \tl_const:Nn \c_@@_standard_tl { standard }
793 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```

794 \bool_new:N \l_@@_light_syntax_bool
```

The string `\l_@@_baseline_tl` may contain one of the three values `t`, `c` or `b` as in the option of the environment `{array}`. However, it may also contain an integer (which represents the number of the row to which align the array).

```

795 \tl_new:N \l_@@_baseline_tl
796 \tl_set:Nn \l_@@_baseline_tl { c }
```

The flag `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (as it is done in `{array}` of `array`).

```

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

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

```

798 \bool_new:N \l_@@_parallelize_diags_bool
799 \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.

```

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

```

801 \dim_new:N \l_@@_notes_above_space_dim
802 \hook_gput_code:nnn { begindocument } { . }
803 { \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.).

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

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

```
805 \cs_new_protected:Npn \@@_reset_arraystretch:
806 { \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).

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

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

```
808 \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.

```
809 \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”.

```
810 \bool_new:N \l_@@_medium_nodes_bool
811 \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).

```
812 \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`).

```
813 \dim_new:N \l_@@_left_margin_dim
814 \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`.

```
815 \dim_new:N \l_@@_extra_left_margin_dim
816 \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.

```
817 \tl_new:N \l_@@_end_of_row_tl
818 \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 “:”.

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

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

```
820 \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.

```

821 \bool_new:N \l_@@_delimiters_max_width_bool

822 \keys_define:nn { NiceMatrix / xdots }
823 {
824   shorten-start .code:n =
825     \hook_gput_code:nnn { begindocument } { . }
826     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
827   shorten-end .code:n =
828     \hook_gput_code:nnn { begindocument } { . }
829     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
830   shorten-start .value_required:n = true ,
831   shorten-end .value_required:n = true ,
832   shorten .code:n =
833     \hook_gput_code:nnn { begindocument } { . }
834     {
835       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
836       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
837     } ,
838   shorten .value_required:n = true ,
839   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
840   horizontal-labels .default:n = true ,
841   line-style .code:n =
842     {
843       \bool_lazy_or:nnTF
844         { \cs_if_exist_p:N \tikzpicture }
845         { \str_if_eq_p:nn { #1 } { standard } }
846         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
847         { \@@_error:n { bad-option-for-line-style } }
848     } ,
849   line-style .value_required:n = true ,
850   color .tl_set:N = \l_@@_xdots_color_tl ,
851   color .value_required:n = true ,
852   radius .code:n =
853     \hook_gput_code:nnn { begindocument } { . }
854     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
855   radius .value_required:n = true ,
856   inter .code:n =
857     \hook_gput_code:nnn { begindocument } { . }
858     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
859   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 a absent `^{\dots}`.

```

860   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
861   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
862   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).

```

863   draw-first .code:n = \prg_do_nothing: ,
864   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
865 }

```

```

866 \keys_define:nn { NiceMatrix / rules }
867 {
868   color .tl_set:N = \l_@@_rules_color_tl ,
869   color .value_required:n = true ,
870   width .dim_set:N = \arrayrulewidth ,
871   width .value_required:n = true ,
872   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
873 }

```

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

```

874 \keys_define:nn { NiceMatrix / Global }
875 {
876   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
877   rounded-corners .default:n = 4 pt ,
878   custom-line .code:n = \@@_custom_line:n { #1 } ,
879   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
880   rules .value_required:n = true ,
881   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
882   standard-cline .default:n = true ,
883   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
884   cell-space-top-limit .value_required:n = true ,
885   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
886   cell-space-bottom-limit .value_required:n = true ,
887   cell-space-limits .meta:n =
888   {
889     cell-space-top-limit = #1 ,
890     cell-space-bottom-limit = #1 ,
891   } ,
892   cell-space-limits .value_required:n = true ,
893   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
894   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
895   light-syntax .default: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   vlimes .clist_set:N = \l_@@_vlimes_clist ,
912   hlines .default:n = all ,
913   vlimes .default:n = all ,
914   vlimes-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 / Env }
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 =
982   \tl_if_eq:nnTF { #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:Nx \l_tmpa_str { #1 }
990     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
991       { \@@_error:nn { Duplicate~name } { #1 } }
992       { \seq_gput_left:NV \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       \IfPackageLoadedTF { enumitem }
1025         { \setlist* [ tabularnotes ] { #1 } }
1026         { }
1027     }
1028   } ,
1029   enumitem-keys .value_required:n = true ,
1030   enumitem-keys-para .code:n =
1031   {
1032     \hook_gput_code:nnn { begindocument } { . }
1033     {
1034       \IfPackageLoadedTF { enumitem }
1035         { \setlist* [ tabularnotes* ] { #1 } }
1036         { }

```



```

1037     }
1038   } ,
1039   enumitem-keys-para .value_required:n = true ,
1040   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1041   detect-duplicates .default:n = true ,
1042   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1043 }
1044 \keys_define:nn { NiceMatrix / delimiters }
1045 {
1046   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1047   max-width .default:n = true ,
1048   color .tl_set:N = \l_@@_delimiters_color_tl ,
1049   color .value_required:n = true ,
1050 }

```

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

```

1051 \keys_define:nn { NiceMatrix }
1052 {
1053   NiceMatrixOptions .inherit:n =
1054     { NiceMatrix / Global } ,
1055   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1056   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1057   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1058   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1059   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1060   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1061   CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1062   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1063   NiceMatrix .inherit:n =
1064     {
1065       NiceMatrix / Global ,
1066       NiceMatrix / Env ,
1067     } ,
1068   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1069   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1070   NiceTabular .inherit:n =
1071     {
1072       NiceMatrix / Global ,
1073       NiceMatrix / Env
1074     } ,
1075   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1076   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1077   NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1078   NiceArray .inherit:n =
1079     {
1080       NiceMatrix / Global ,
1081       NiceMatrix / Env ,
1082     } ,
1083   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1084   NiceArray / rules .inherit:n = NiceMatrix / rules ,
1085   pNiceArray .inherit:n =
1086     {
1087       NiceMatrix / Global ,
1088       NiceMatrix / Env ,
1089     } ,
1090   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1091   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1092 }

```

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

```

1093 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1094 {
1095   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1096   delimiters / color .value_required:n = true ,
1097   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1098   delimiters / max-width .default:n = true ,
1099   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1100   delimiters .value_required:n = true ,
1101   width .dim_set:N = \l_@@_width_dim ,
1102   width .value_required:n = true ,
1103   last-col .code:n =
1104     \tl_if_empty:nF { #1 }
1105     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1106     \int_zero:N \l_@@_last_col_int ,
1107   small .bool_set:N = \l_@@_small_bool ,
1108   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.

```

1109   renew-matrix .code:n = \@@_renew_matrix: ,
1110   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}`.

```

1111   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.

```

1112   columns-width .code:n =
1113     \tl_if_eq:nnTF { #1 } { auto }
1114     { \@@_error:n { Option-auto-for-columns-width } }
1115     { \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.

```

1116   allow-duplicate-names .code:n =
1117     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1118   allow-duplicate-names .value_forbidden:n = true ,
1119   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1120   notes .value_required:n = true ,
1121   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1122   sub-matrix .value_required:n = true ,
1123   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1124   matrix / columns-type .value_required:n = true ,
1125   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1126   caption-above .default:n = true ,
1127   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1128 }

```

`\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.

```

1129 \NewDocumentCommand \NiceMatrixOptions { m }
1130 { \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.

```

1131 \keys_define:nn { NiceMatrix / NiceMatrix }
1132 {
1133   last-col .code:n = \tl_if_empty:nTF { #1 }

```

```

1134         {
1135             \bool_set_true:N \l_@@_last_col_without_value_bool
1136             \int_set:Nn \l_@@_last_col_int { -1 }
1137         }
1138         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1139     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1140     columns-type .value_required:n = true ,
1141     l .meta:n = { columns-type = l } ,
1142     r .meta:n = { columns-type = r } ,
1143     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1144     delimiters / color .value_required:n = true ,
1145     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1146     delimiters / max-width .default:n = true ,
1147     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1148     delimiters .value_required:n = true ,
1149     small .bool_set:N = \l_@@_small_bool ,
1150     small .value_forbidden:n = true ,
1151     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1152 }

```

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

```

1153 \keys_define:nn { NiceMatrix / NiceArray }
1154 {

```

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.

```

1155     small .bool_set:N = \l_@@_small_bool ,
1156     small .value_forbidden:n = true ,
1157     last-col .code:n = \tl_if_empty:nF { #1 }
1158         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1159         \int_zero:N \l_@@_last_col_int ,
1160     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1161     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1162     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1163 }
1164 \keys_define:nn { NiceMatrix / pNiceArray }
1165 {
1166     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1167     last-col .code:n = \tl_if_empty:nF { #1 }
1168         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1169         \int_zero:N \l_@@_last_col_int ,
1170     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1171     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1172     delimiters / color .value_required:n = true ,
1173     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1174     delimiters / max-width .default:n = true ,
1175     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1176     delimiters .value_required:n = true ,
1177     small .bool_set:N = \l_@@_small_bool ,
1178     small .value_forbidden:n = true ,
1179     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1180     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1181     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1182 }

```

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

```

1183 \keys_define:nn { NiceMatrix / NiceTabular }
1184 {

```

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.

```

1185     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1186           \bool_set_true:N \l_@@_width_used_bool ,
1187     width .value_required:n = true ,
1188     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1189     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1190     tabularnote .value_required:n = true ,
1191     caption .tl_set:N = \l_@@_caption_tl ,
1192     caption .value_required:n = true ,
1193     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1194     short-caption .value_required:n = true ,
1195     label .tl_set:N = \l_@@_label_tl ,
1196     label .value_required:n = true ,
1197     last-col .code:n = \tl_if_empty:nF {#1}
1198           { \@@_error:n { last-col-non-empty-for-NiceArray } }
1199           \int_zero:N \l_@@_last_col_int ,
1200     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1201     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1202     unknown .code:n = \@@_error:n { Unknown~key~for-NiceTabular }
1203   }

```

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`

```

1204 \keys_define:nn { NiceMatrix / CodeAfter }
1205 {
1206     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1207     delimiters / color .value_required:n = true ,
1208     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1209     rules .value_required:n = true ,
1210     xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1211     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1212     sub-matrix .value_required:n = true ,
1213     unknown .code:n = \@@_error:n { Unknown~key~for-CodeAfter }
1214 }

```

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}`).

```

1215 \cs_new_protected:Npn \@@_cell_begin:w
1216 {

```

`\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.

```

1217     \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).

```

1218     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

1219     \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.

```
1220 \int_compare:nNnT \c@jCol = \c_one_int
1221 { \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:`.

```
1222 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1223 \@@_tuning_not_tabular_begin:
1224 \@@_tuning_first_row:
1225 \@@_tuning_last_row:
1226 \g_@@_row_style_tl
1227 }
```

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

```
1228 \cs_new_protected:Npn \@@_tuning_first_row:
1229 {
1230   \int_if_zero:nT \c@iRow
1231   {
1232     \int_compare:nNnT \c@jCol > \c_zero_int
1233     {
1234       \l_@@_code_for_first_row_tl
1235       \xglobal \colorlet { nicematrix-first-row } { . }
1236     }
1237   }
1238 }
```

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

```
1239 \cs_new_protected:Npn \@@_tuning_last_row:
1240 {
1241   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1242   {
1243     \l_@@_code_for_last_row_tl
1244     \xglobal \colorlet { nicematrix-last-row } { . }
1245   }
1246 }
```

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

```
1247 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1248 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1249 {
1250   \c_math_toggle_token
```

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

```
1251 \@@_tuning_key_small:
1252 }
1253 \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.

```
1254 \cs_new_protected:Npn \@@_begin_of_row:
1255 {
1256   \int_gincr:N \c@iRow
1257   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1258   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1259   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1260   \pgfpicture
1261   \pgfrememberpicturepositiononpagetrue
```

```

1262 \pgfcoordinate
1263 { \@@_env: - row - \int_use:N \c@iRow - base }
1264 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1265 \str_if_empty:NF \l_@@_name_str
1266 {
1267   \pgfnodealias
1268   { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1269   { \@@_env: - row - \int_use:N \c@iRow - base }
1270 }
1271 \endpgfpicture
1272 }

```

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.

```

1273 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1274 {
1275   \int_if_zero:nTF \c@iRow
1276   {
1277     \dim_gset:Nn \g_@@_dp_row_zero_dim
1278     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1279     \dim_gset:Nn \g_@@_ht_row_zero_dim
1280     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1281   }
1282   {
1283     \int_compare:nNnT \c@iRow = \c_one_int
1284     {
1285       \dim_gset:Nn \g_@@_ht_row_one_dim
1286       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1287     }
1288   }
1289 }
1290 \cs_new_protected:Npn \@@_rotate_cell_box:
1291 {
1292   \box_rotate:Nn \l_@@_cell_box { 90 }
1293   \bool_if:NTF \g_@@_rotate_c_bool
1294   {
1295     \hbox_set:Nn \l_@@_cell_box
1296     {
1297       \c_math_toggle_token
1298       \vcenter { \box_use:N \l_@@_cell_box }
1299       \c_math_toggle_token
1300     }
1301   }
1302   {
1303     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1304     {
1305       \vbox_set_top:Nn \l_@@_cell_box
1306       {
1307         \vbox_to_zero:n { }
1308         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1309         \box_use:N \l_@@_cell_box
1310       }
1311     }
1312   }
1313   \bool_gset_false:N \g_@@_rotate_bool
1314   \bool_gset_false:N \g_@@_rotate_c_bool
1315 }
1316 \cs_new_protected:Npn \@@_adjust_size_box:

```

```

1317 {
1318   \dim_compare:nNtT \g_@@_blocks_wd_dim > \c_zero_dim
1319   {
1320     \box_set_wd:Nn \l_@@_cell_box
1321     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1322     \dim_gzero:N \g_@@_blocks_wd_dim
1323   }
1324   \dim_compare:nNtT \g_@@_blocks_dp_dim > \c_zero_dim
1325   {
1326     \box_set_dp:Nn \l_@@_cell_box
1327     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1328     \dim_gzero:N \g_@@_blocks_dp_dim
1329   }
1330   \dim_compare:nNtT \g_@@_blocks_ht_dim > \c_zero_dim
1331   {
1332     \box_set_ht:Nn \l_@@_cell_box
1333     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1334     \dim_gzero:N \g_@@_blocks_ht_dim
1335   }
1336 }
1337 \cs_new_protected:Npn \@@_cell_end:
1338 {

```

The following command is nullified in the tabulars.

```

1339   \@@_tuning_not_tabular_end:
1340   \hbox_set_end:
1341   \@@_cell_end_i:
1342 }
1343 \cs_new_protected:Npn \@@_cell_end_i:
1344 {

```

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.

```

1345   \g_@@_cell_after_hook_tl
1346   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1347   \@@_adjust_size_box:
1348   \box_set_ht:Nn \l_@@_cell_box
1349   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1350   \box_set_dp:Nn \l_@@_cell_box
1351   { \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”).

```

1352   \@@_update_max_cell_width:

```

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

```

1353   \@@_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.

```

1354 \bool_if:NTF \g_@@_empty_cell_bool
1355 { \box_use_drop:N \l_@@_cell_box }
1356 {
1357   \bool_if:NTF
1358     \g_@@_not_empty_cell_bool
1359     \@@_node_for_cell:
1360     {
1361       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1362       \@@_node_for_cell:
1363       { \box_use_drop:N \l_@@_cell_box }
1364     }
1365   }
1366   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1367   \bool_gset_false:N \g_@@_empty_cell_bool
1368   \bool_gset_false:N \g_@@_not_empty_cell_bool
1369 }

```

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

```

1370 \cs_new_protected:Npn \@@_update_max_cell_width:
1371 {
1372   \dim_gset:Nn \g_@@_max_cell_width_dim
1373   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1374 }

```

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`).

```

1375 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1376 {
1377   \@@_math_toggle:
1378   \hbox_set_end:
1379   \bool_if:NF \g_@@_rotate_bool
1380   {
1381     \hbox_set:Nn \l_@@_cell_box
1382     {
1383       \makebox [ \l_@@_col_width_dim ] [ s ]
1384       { \hbox_unpack_drop:N \l_@@_cell_box }
1385     }
1386   }
1387   \@@_cell_end_i:
1388 }

```

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

```

1389 \pgfset
1390 {
1391   nicematrix / cell-node /.style =
1392   {
1393     inner-sep = \c_zero_dim ,
1394     minimum-width = \c_zero_dim
1395   }
1396 }
1397 \cs_new_protected:Npn \@@_node_for_cell:
1398 {
1399   \pgfpicture
1400   \pgfsetbaseline \c_zero_dim

```



```

1401 \pgfrememberpicturepositiononpagetrue
1402 \pgfset { nicematrix / cell-node }
1403 \pgfnode
1404 { rectangle }
1405 { base }
1406 {

```

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

```

1407 \set@color
1408 \box_use_drop:N \l_@@_cell_box
1409 }
1410 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1411 { \l_@@_pgf_node_code_tl }
1412 \str_if_empty:NF \l_@@_name_str
1413 {
1414 \pgfnodealias
1415 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1416 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1417 }
1418 \endpgfpicture
1419 }

```

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.

```

1420 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1421 {
1422 \cs_new_protected:Npn \@@_patch_node_for_cell:
1423 {
1424 \hbox_set:Nn \l_@@_cell_box
1425 {
1426 \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1427 \hbox_overlap_left:n
1428 {
1429 \pgfsys@markposition
1430 { \@@_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.

```

1431 #1
1432 }
1433 \box_use:N \l_@@_cell_box
1434 \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1435 \hbox_overlap_left:n
1436 {
1437 \pgfsys@markposition
1438 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1439 #1
1440 }
1441 }
1442 }
1443 }

```

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

```

1444 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1445 {
1446 \@@_patch_node_for_cell:n
1447 { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1448 }
1449 { \@@_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).

```
1450 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1451 {
1452   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1453   { \g_@@_#2 _ lines _ tl }
1454   {
1455     \use:c { @@ _ draw _ #2 : nnn }
1456     { \int_use:N \c@iRow }
1457     { \int_use:N \c@jCol }
1458     { \exp_not:n { #3 } }
1459   }
1460 }
```

```
1461 \cs_new_protected:Npn \@@_array:
1462 {
```

The following line is only a speed-up: it's a redefinition of `\@mkpream` of `array` in order to speed up the compilation by deleting one line of code in `\@mkpream` (the expansion of the preamble). In the classes of REVTeX, that command `\@@_redefine_mkpream:` will be nullified (no speed-up).

```
1463 \@@_redefine_mkpream:
1464 \dim_set:Nn \col@sep
1465 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1466 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1467 { \cs_set_nopar:Npn \@halignto { } }
1468 { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1469 \@tabarray
```

`\l_@@_baseline_tl` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of `array`) with the option `t` and the right translation will be done further. Remark that `\str_if_eq:VnTF` is fully expandable and we need something fully expandable here.

```
1470 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1471 }
```

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.

```
1472 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1473 \cs_new_protected:Npn \@@_create_row_node:
1474 {
1475   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1476   {
1477     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
```

```

1478     \@@_create_row_node_i:
1479   }
1480 }

1481 \cs_new_protected:Npn \@@_create_row_node_i:
1482 {

```

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

```

1483   \hbox
1484   {
1485     \bool_if:NT \l_@@_code_before_bool
1486     {
1487       \vtop
1488       {
1489         \skip_vertical:N 0.5\arrayrulewidth
1490         \pgfsys@markposition
1491         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1492         \skip_vertical:N -0.5\arrayrulewidth
1493       }
1494     }
1495     \pgfpicture
1496     \pgfrememberpicturepositiononpagetrue
1497     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1498     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1499     \str_if_empty:NF \l_@@_name_str
1500     {
1501       \pgfnodealias
1502       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1503       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1504     }
1505     \endpgfpicture
1506   }
1507 }

```

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

```

1508 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1509 \cs_new_protected:Npn \@@_everycr_i:
1510 {
1511   \int_gzero:N \c@jCol
1512   \bool_gset_false:N \g_@@_after_col_zero_bool
1513   \bool_if:NF \g_@@_row_of_col_done_bool
1514   {
1515     \@@_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).

```

1516     \tl_if_empty:NF \l_@@_hlines_clist
1517     {
1518       \tl_if_eq:NnF \l_@@_hlines_clist \c_@@_all_tl
1519       {
1520         \exp_args:NNe
1521         \clist_if_in:NnT
1522         \l_@@_hlines_clist
1523         { \int_eval:n { \c@iRow + 1 } }
1524       }
1525     }

```

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.

```

1526     \int_compare:nNnT \c@iRow > { -1 }
1527     {
1528       \int_compare:nNnF \c@iRow = \l_@@_last_row_int

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. The package `nicematrix` uses it even if `colortbl` is not loaded. We use a TeX group in order to limit the scope of `\CT@arc@`.

```

1529             { \hrule height \arrayrulewidth width \c_zero_dim }
1530         }
1531     }
1532 }
1533 }
1534 }
```

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

```

1535 \cs_set_protected:Npn \@@_renew_dots:
1536 {
1537     \cs_set_eq:NN \ldots \@@_Ldots
1538     \cs_set_eq:NN \cdots \@@_Cdots
1539     \cs_set_eq:NN \vdots \@@_Vdots
1540     \cs_set_eq:NN \ddots \@@_Ddots
1541     \cs_set_eq:NN \iddots \@@_Iddots
1542     \cs_set_eq:NN \dots \@@_Ldots
1543     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1544 }
1545 \cs_new_protected:Npn \@@_test_color_inside:
1546 {
1547     \bool_if:NF \l_@@_color_inside_bool
1548     {
```

We will issue an error only during the first run.

```

1549         \bool_if:NF \g_@@_aux_found_bool
1550         { \@@_error:n { without~color~inside } }
1551     }
1552 }
```

```

1553 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1554 \hook_gput_code:nnn { begindocument } { . }
1555 {
1556     \IfPackageLoadedTF { colortbl }
1557     {
1558         \cs_set_protected:Npn \@@_redefine_everycr:
1559         {
1560             \CT@everycr
1561             {
1562                 \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1563                 \@@_everycr:
1564             }
1565         }
1566     }
1567 } }
```

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.

```

1569 \hook_gput_code:nnn { begindocument } { . }
1570 {
```

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

```

1571 \IfPackageLoadedTF { booktabs }
1572 {
1573   \cs_new_protected:Npn \@@_patch_booktabs:
1574   { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1575 }
1576 { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1577 }

```

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

```

1578 \cs_new_protected:Npn \@@_pre_array_ii:
1579 {

```

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

```

1580   \int_gzero:N \g_@@_total_X_weight_int
1581   \@@_expand_clist:N \l_@@_hlines_clist
1582   \@@_expand_clist:N \l_@@_vlines_clist
1583   \@@_patch_booktabs:
1584   \box_clear_new:N \l_@@_cell_box
1585   \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}`).

```

1586   \bool_if:NT \l_@@_small_bool
1587   {
1588     \cs_set_nopar:Npn \arraystretch { 0.47 }
1589     \dim_set:Nn \arraycolsep { 1.45 pt }

```

By default, `\@@_small_scripstyle:` is null.

```

1590     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1591   }

```

```

1592   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1593   {
1594     \tl_put_right:Nn \@@_begin_of_row:
1595     {
1596       \pgfsys@markposition
1597       { \@@_env: - row - \int_use:N \c@iRow - base }
1598     }
1599   }

```

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`.

```

1600   \cs_set_nopar:Npn \ialign
1601   {
1602     \@@_redefine_everycr:
1603     \tabskip = \c_zero_skip

```

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

```

1604     \dim_gzero_new:N \g_@@_dp_row_zero_dim

```

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

```

1605 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1606 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1607 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1608 \dim_gzero_new:N \g_@@_ht_row_one_dim
1609 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1610 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1611 \dim_gzero_new:N \g_@@_ht_last_row_dim
1612 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1613 \dim_gzero_new:N \g_@@_dp_last_row_dim
1614 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }

```

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

```

1615 \cs_set_eq:NN \ialign \@@_old_ialign:
1616 \halign
1617 }

```

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).

```

1618 \cs_set_eq:NN \@@_old_ldots \ldots
1619 \cs_set_eq:NN \@@_old_cdots \cdots
1620 \cs_set_eq:NN \@@_old_vdots \vdots
1621 \cs_set_eq:NN \@@_old_ddots \ddots
1622 \cs_set_eq:NN \@@_old_iddots \iddots
1623 \bool_if:NTF \l_@@_standard_cline_bool
1624 { \cs_set_eq:NN \cline \@@_standard_cline }
1625 { \cs_set_eq:NN \cline \@@_cline }
1626 \cs_set_eq:NN \Ldots \@@_Ldots
1627 \cs_set_eq:NN \Cdots \@@_Cdots
1628 \cs_set_eq:NN \Vdots \@@_Vdots
1629 \cs_set_eq:NN \Ddots \@@_Ddots
1630 \cs_set_eq:NN \Iddots \@@_Iddots
1631 \cs_set_eq:NN \Hline \@@_Hline:
1632 \cs_set_eq:NN \Hspace \@@_Hspace:
1633 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1634 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1635 \cs_set_eq:NN \Block \@@_Block:
1636 \cs_set_eq:NN \rotate \@@_rotate:
1637 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1638 \cs_set_eq:NN \dotfill \@@_dotfill:
1639 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1640 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1641 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1642 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1643 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1644 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1645 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1646 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1647 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1648 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1649 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1650 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1651 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1652 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1653 \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.

```

1654 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1655 \hook_gput_code:nnn { env / tabular / begin } { . }
1656 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1657 \@@_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!).

```

1658 \tl_if_exist:NT \l_@@_note_in_caption_tl
1659 {
1660   \tl_if_empty:NF \l_@@_note_in_caption_tl
1661   {
1662     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1663     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1664   }
1665 }

```

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).

```

1666 \seq_gclear:N \g_@@_multicolumn_cells_seq
1667 \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).

```

1668 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1669 \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.

```

1670 \int_gzero_new:N \g_@@_col_total_int
1671 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1672 \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.

```

1673 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1674 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1675 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1676 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1677 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1678 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1679 \tl_gclear:N \g_nicematrix_code_before_tl
1680 \tl_gclear:N \g_@@_pre_code_before_tl
1681 }

```

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

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

```

1682 \cs_new_protected:Npn \@@_pre_array:
1683 {
1684   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1685   \int_gzero_new:N \c@iRow
1686   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1687   \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).

```

1688   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1689   {
1690     \bool_set_true:N \l_@@_last_row_without_value_bool
1691     \bool_if:NT \g_@@_aux_found_bool
1692       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1693   }
1694   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1695   {
1696     \bool_if:NT \g_@@_aux_found_bool
1697       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1698   }

```

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”.

```

1699   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1700   {
1701     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1702       {
1703         \dim_gset:Nn \g_@@_ht_last_row_dim
1704           { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1705         \dim_gset:Nn \g_@@_dp_last_row_dim
1706           { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1707       }
1708   }

1709   \seq_gclear:N \g_@@_cols_vlism_seq
1710   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1711   \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.

```

1712   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1713   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1714   \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.

```

1715   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1716   \@@_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.

```
1717 \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]`).

```
1718 \dim_zero_new:N \l_@@_left_delim_dim
1719 \dim_zero_new:N \l_@@_right_delim_dim
1720 \bool_if:NTF \g_@@_delims_bool
1721 {
```

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

```
1722 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1723 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1724 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1725 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1726 }
1727 {
1728 \dim_gset:Nn \l_@@_left_delim_dim
1729 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1730 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1731 }
```

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).

```
1732 \hbox_set:Nw \l_@@_the_array_box
1733 \skip_horizontal:N \l_@@_left_margin_dim
1734 \skip_horizontal:N \l_@@_extra_left_margin_dim
1735 \c_math_toggle_token
1736 \bool_if:NTF \l_@@_light_syntax_bool
1737 { \use:c { @@-light-syntax } }
1738 { \use:c { @@-normal-syntax } }
1739 }
```

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

```
1740 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1741 {
1742 \tl_set:Nn \l_tmpa_tl { #1 }
1743 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1744 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1745 \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1746 \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...

```
1747 \@@_pre_array:
1748 }
```

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 only for legibility).

```
1749 \cs_new_protected:Npn \@@_pre_code_before:
1750 {
```

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).

```

1751 \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1752 \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1753 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1754 \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.

```

1755 \pgfsys@markposition { \@@_env: - position }
1756 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1757 \pgfpicture
1758 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1759 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1760 {
1761   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1762   \pgfcoordinate { \@@_env: - row - ##1 }
1763   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1764 }

```

Now, the recreation of the `col` nodes.

```

1765 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1766 {
1767   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1768   \pgfcoordinate { \@@_env: - col - ##1 }
1769   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1770 }

```

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

```

1771 \@@_create_diag_nodes:

```

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

```

1772 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1773 \endpgfpicture

```

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

```

1774 \@@_create_blocks_nodes:
1775 \IfPackageLoadedTF { tikz }
1776 {
1777   \tikzset
1778   {
1779     every-picture / .style =
1780     { overlay , name~prefix = \@@_env: - }
1781   }
1782 }
1783 { }
1784 \cs_set_eq:NN \cellcolor \@@_cellcolor
1785 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1786 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1787 \cs_set_eq:NN \rowcolor \@@_rowcolor
1788 \cs_set_eq:NN \rowcolors \@@_rowcolors
1789 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1790 \cs_set_eq:NN \arraycolor \@@_arraycolor
1791 \cs_set_eq:NN \columncolor \@@_columncolor
1792 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1793 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1794 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1795 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1796 }

```

```

1797 \cs_new_protected:Npn \@@_exec_code_before:
1798 {
1799   \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`.

```

1800   \@@_add_to_colors_seq:nn { { nocolor } } { }
1801   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1802   \group_begin:

```

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

```

1803   \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`).

```

1804   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1805   { \@@_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`.

```

1806   \exp_last_unbraced:NV \@@_CodeBefore_keys:
1807   \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.

```

1808   \@@_actually_color:
1809   \l_@@_code_before_tl
1810   \q_stop
1811   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1812   \group_end:
1813   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1814   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1815 }

```

```

1816 \keys_define:nn { NiceMatrix / CodeBefore }
1817 {
1818   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1819   create-cell-nodes .default:n = true ,
1820   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1821   sub-matrix .value_required:n = true ,
1822   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1823   delimiters / color .value_required:n = true ,
1824   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1825 }

1826 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1827 {
1828   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1829   \@@_CodeBefore:w
1830 }

```

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.

```

1831 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1832 {

```

```

1833 \bool_if:NT \g_@@_aux_found_bool
1834 {
1835     \@@_pre_code_before:
1836     #1
1837 }
1838 }

```

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.

```

1839 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1840 {
1841     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1842     {
1843         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1844         \pgfcoordinate { \@@_env: - row - ##1 - base }
1845         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1846         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1847         {
1848             \cs_if_exist:cT
1849             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1850             {
1851                 \pgfsys@getposition
1852                 { \@@_env: - ##1 - #####1 - NW }
1853                 \@@_node_position:
1854                 \pgfsys@getposition
1855                 { \@@_env: - ##1 - #####1 - SE }
1856                 \@@_node_position_i:
1857                 \@@_pgf_rect_node:nnn
1858                 { \@@_env: - ##1 - #####1 }
1859                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1860                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1861             }
1862         }
1863     }
1864     \int_step_inline:nn \c@iRow
1865     {
1866         \pgfnodealias
1867         { \@@_env: - ##1 - last }
1868         { \@@_env: - ##1 - \int_use:N \c@jCol }
1869     }
1870     \int_step_inline:nn \c@jCol
1871     {
1872         \pgfnodealias
1873         { \@@_env: - last - ##1 }
1874         { \@@_env: - \int_use:N \c@iRow - ##1 }
1875     }
1876     \@@_create_extra_nodes:
1877 }

1878 \cs_new_protected:Npn \@@_create_blocks_nodes:
1879 {
1880     \pgfpicture
1881     \pgf@relevantforpicturesizefalse
1882     \pgfrememberpicturepositiononpagetrue
1883     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1884     { \@@_create_one_block_node:nnnnn ##1 }
1885     \endpgfpicture
1886 }

```

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

```

1887 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1888 {
1889   \tl_if_empty:nF { #5 }
1890   {
1891     \@@_qpoint:n { col - #2 }
1892     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1893     \@@_qpoint:n { #1 }
1894     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1895     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1896     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1897     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1898     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1899     \@@_pgf_rect_node:nnnnn
1900     { \@@_env: - #5 }
1901     { \dim_use:N \l_tmpa_dim }
1902     { \dim_use:N \l_tmpb_dim }
1903     { \dim_use:N \l_@@_tmpc_dim }
1904     { \dim_use:N \l_@@_tmpd_dim }
1905   }
1906 }

1907 \cs_new_protected:Npn \@@_patch_for_revtext:
1908 {
1909   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1910   \cs_set_eq:NN \insert@column \insert@column@array
1911   \cs_set_eq:NN \@classx \@classx@array
1912   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1913   \cs_set_eq:NN \@arraycr \@arraycr@array
1914   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1915   \cs_set_eq:NN \array \array@array
1916   \cs_set_eq:NN \@array \@array@array
1917   \cs_set_eq:NN \@tabular \@tabular@array
1918   \cs_set_eq:NN \@mkpream \@mkpream@array
1919   \cs_set_eq:NN \endarray \endarray@array
1920   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1921   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1922 }

```

11 The environment `{NiceArrayWithDelims}`

```

1923 \NewDocumentEnvironment { NiceArrayWithDelims }
1924 { m m O { } m ! O { } t \CodeBefore }
1925 {
1926   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1927   \@@_provide_pgfsyspdfmark:
1928   \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.

```

1929   \bgroup

1930   \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).

```

1931 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1932 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1933 \int_gzero:N \g_@@_block_box_int
1934 \dim_zero:N \g_@@_width_last_col_dim
1935 \dim_zero:N \g_@@_width_first_col_dim
1936 \bool_gset_false:N \g_@@_row_of_col_done_bool
1937 \str_if_empty:NT \g_@@_name_env_str
1938 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1939 \bool_if:NTF \l_@@_tabular_bool
1940 \mode_leave_vertical:
1941 \@@_test_if_math_mode:
1942 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1943 \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.

```

1944 \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.

```

1945 \cs_if_exist:NT \tikz@library@external@loaded
1946 {
1947   \tikzexternaldisable
1948   \cs_if_exist:NT \ifstandalone
1949     { \tikzset { external / optimize = false } }
1950 }

```

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

```

1951 \int_gincr:N \g_@@_env_int
1952 \bool_if:NF \l_@@_block_auto_columns_width_bool
1953 { \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`).

```

1954 \seq_gclear:N \g_@@_blocks_seq
1955 \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`.

```

1956 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1957 \seq_gclear:N \g_@@_pos_of_xdots_seq
1958 \tl_gclear_new:N \g_@@_code_before_tl
1959 \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.

```

1960 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1961 {
1962   \bool_gset_true:N \g_@@_aux_found_bool
1963   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1964 }
1965 { \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.

```

1966 \tl_build_gbegin:N \g_@@_aux_tl

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1967 \tl_if_empty:NF \g_@@_code_before_tl
1968 {
1969     \bool_set_true:N \l_@@_code_before_bool
1970     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1971 }
1972 \tl_if_empty:NF \g_@@_pre_code_before_tl
1973 { \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.

```

1974 \bool_if:NTF \g_@@_delims_bool
1975 { \keys_set:nn { NiceMatrix / pNiceArray } }
1976 { \keys_set:nn { NiceMatrix / NiceArray } }
1977 { #3 , #5 }

```

```

1978 \@@_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:.

```

1979 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1980 }

```

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

```

1981 {
1982     \bool_if:NTF \l_@@_light_syntax_bool
1983     { \use:c { end @@-light-syntax } }
1984     { \use:c { end @@-normal-syntax } }
1985     \c_math_toggle_token
1986     \skip_horizontal:N \l_@@_right_margin_dim
1987     \skip_horizontal:N \l_@@_extra_right_margin_dim
1988     \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.

```

1989 \bool_if:NT \l_@@_width_used_bool
1990 {
1991     \int_if_zero:nT \g_@@_total_X_weight_int
1992     { \@@_error_or_warning:n { width-without-X-columns } }
1993 }

```

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 .

```

1994 \int_compare:nNtT \g_@@_total_X_weight_int > \c_zero_int
1995 {
1996     \tl_build_gput_right:Nx \g_@@_aux_tl
1997     {
1998         \bool_set_true:N \l_@@_X_columns_aux_bool
1999         \dim_set:Nn \l_@@_X_columns_dim
2000         {
2001             \dim_compare:nNnTF
2002             {
2003                 \dim_abs:n
2004                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2005             }
2006             <
2007             { 0.001 pt }

```

```

2008         { \dim_use:N \l_@@_X_columns_dim }
2009         {
2010             \dim_eval:n
2011             {
2012                 ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2013                 / \int_use:N \g_@@_total_X_weight_int
2014                 + \l_@@_X_columns_dim
2015             }
2016         }
2017     }
2018 }
2019 }

```

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).

```

2020 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2021 {
2022     \bool_if:NF \l_@@_last_row_without_value_bool
2023     {
2024         \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2025         {
2026             \@@_error:n { Wrong-last-row }
2027             \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2028         }
2029     }
2030 }

```

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

```

2031 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2032 \bool_if:NnTF \g_@@_last_col_found_bool
2033 { \int_gdecr:N \c@jCol }
2034 {
2035     \int_compare:nNnT \l_@@_last_col_int > { -1 }
2036     { \@@_error:n { last-col-not-used } }
2037 }

```

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

```

2038 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2039 \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. 89).

```

2040 \int_if_zero:nT \l_@@_first_col_int
2041 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2042 \bool_if:nTF { ! \g_@@_delims_bool }
2043 {
2044     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2045     \@@_use_arraybox_with_notes_c:
2046     {
2047         \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2048         \@@_use_arraybox_with_notes_b:
2049         \@@_use_arraybox_with_notes:
2050     }
2051 }

```

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).

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


```

2052 {
2053   \int_if_zero:nTF \l_@@_first_row_int
2054   {
2055     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2056     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2057   }
2058   { \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

```

2059   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2060   {
2061     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2062     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2063   }
2064   { \dim_zero:N \l_tmpb_dim }
2065   \hbox_set:Nn \l_tmpa_box
2066   {
2067     \c_math_toggle_token
2068     \@@_color:o \l_@@_delimiters_color_tl
2069     \exp_after:wN \left \g_@@_left_delim_tl
2070     \vcenter
2071     {

```

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.

```

2072       \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2073       \hbox
2074       {
2075         \bool_if:NTF \l_@@_tabular_bool
2076         { \skip_horizontal:N -\tabcolsep }
2077         { \skip_horizontal:N -\arraycolsep }
2078         \@@_use_arraybox_with_notes_c:
2079         \bool_if:NTF \l_@@_tabular_bool
2080         { \skip_horizontal:N -\tabcolsep }
2081         { \skip_horizontal:N -\arraycolsep }
2082       }

```

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

```

2083       \skip_vertical:N -\l_tmpb_dim
2084       \skip_vertical:N \arrayrulewidth
2085     }
2086     \exp_after:wN \right \g_@@_right_delim_tl
2087     \c_math_toggle_token
2088   }

```

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

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

```

2089   \bool_if:NTF \l_@@_delimiters_max_width_bool
2090   {
2091     \@@_put_box_in_flow_bis:nn
2092     \g_@@_left_delim_tl
2093     \g_@@_right_delim_tl
2094   }
2095   \@@_put_box_in_flow:
2096 }

```

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. 90).

```

2097   \bool_if:NT \g_@@_last_col_found_bool
2098   { \skip_horizontal:N \g_@@_width_last_col_dim }

```

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).

```

2099 \bool_if:NT \l_@@_preamble_bool
2100 {
2101   \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2102   { \@@_warning_gredirect_none:n { columns~not~used } }
2103 }
2104 \@@_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.

```

2105 \egroup

```

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

```

2106 \tl_build_gend:N \g_@@_aux_tl
2107 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2108 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2109 \iow_now:Nx \@mainaux
2110 {
2111   \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2112   { \exp_not:o \g_@@_aux_tl }
2113 }
2114 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2115 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2116 }

```

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 that 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.

```

2117 \cs_new_protected:Npn \@@_transform_preamble:
2118 {
2119   \@@_transform_preamble_i:
2120   \@@_transform_preamble_ii:
2121 }

2122 \cs_new_protected:Npn \@@_transform_preamble_i:
2123 {
2124   \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`).

```

2125 \seq_gclear:N \g_@@_cols_vlsim_seq

```

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

```

2126 \bool_gset_false:N \g_tmpb_bool

```

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

```

2127 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2128 \int_zero:N \l_tmpa_int
2129 \tl_gclear:N \g_@@_array_preamble_tl
2130 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2131 {
2132   \tl_gset:Nn \g_@@_array_preamble_tl
2133   { ! { \skip_horizontal:N \arrayrulewidth } }

```

```

2134     }
2135     {
2136         \clist_if_in:NnT \l_@@_vlines_clist 1
2137         {
2138             \tl_gset:Nn \g_@@_array_preamble_tl
2139                 { ! { \skip_horizontal:N \arrayrulewidth } }
2140         }
2141     }

```

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

```

2142     \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2143     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2144     \@@_replace_columncolor:
2145 }

```

```

2146 \hook_gput_code:nnn { begindocument } { . }
2147 {
2148     \IfPackageLoadedTF { colortbl }
2149     {
2150         \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2151         \cs_new_protected:Npn \@@_replace_columncolor:
2152         {
2153             \regex_replace_all:NnN
2154                 \c_@@_columncolor_regex
2155                 { \c { @@_columncolor_preamble } }
2156                 \g_@@_array_preamble_tl
2157         }
2158     }
2159     {
2160         \cs_new_protected:Npn \@@_replace_columncolor:
2161             { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2162     }
2163 }

```

```

2164 \cs_new_protected:Npn \@@_transform_preamble_ii:
2165 {

```

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

```

2166     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2167     {
2168         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2169         { \bool_gset_true:N \g_@@_delims_bool }
2170     }
2171     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2172     \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).

```

2173     \int_if_zero:nTF \l_@@_first_col_int
2174     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2175     {
2176         \bool_if:NF \g_@@_delims_bool
2177         {
2178             \bool_if:NF \l_@@_tabular_bool
2179             {
2180                 \tl_if_empty:NT \l_@@_vlines_clist

```

```

2181         {
2182             \bool_if:NF \l_@@_exterior_arraycolsep_bool
2183             { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2184         }
2185     }
2186 }
2187
2188 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2189 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2190 {
2191     \bool_if:NF \g_@@_delims_bool
2192     {
2193         \bool_if:NF \l_@@_tabular_bool
2194         {
2195             \tl_if_empty:NT \l_@@_vlines_clist
2196             {
2197                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2198                 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2199             }
2200         }
2201     }
2202 }

```

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`).

```

2203 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2204 {
2205     \tl_gput_right:Nn \g_@@_array_preamble_tl
2206     { > { \@@_error_too_much_cols: } 1 }
2207 }
2208 }

```

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.

```

2209 \cs_new_protected:Npn \@@_rec_preamble:n #1
2210 {

```

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

```

2211 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2212 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2213 {

```

Now, the columns defined by `\newcolumntype` of array.

```

2214 \cs_if_exist:cTF { NC @ find @ #1 }
2215 {
2216     \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2217     \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2218 }
2219 {
2220     \tl_if_eq:nnT { #1 } { S }
2221     { \@@_fatal:n { unknown~column~type~S } }
2222     { \@@_fatal:nn { unknown~column~type } { #1 } }
2223 }
2224 }
2225 }

```

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`.

For c, l and r

```

2226 \cs_new:Npn \@@_c #1
2227 {
2228   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2229   \tl_gclear:N \g_@@_pre_cell_tl
2230   \tl_gput_right:Nn \g_@@_array_preamble_tl
2231     { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2232   \int_gincr:N \c@jCol
2233   \@@_rec_preamble_after_col:n
2234 }

2235 \cs_new:Npn \@@_l #1
2236 {
2237   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2238   \tl_gclear:N \g_@@_pre_cell_tl
2239   \tl_gput_right:Nn \g_@@_array_preamble_tl
2240     {
2241       > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2242       l
2243       < \@@_cell_end:
2244     }
2245   \int_gincr:N \c@jCol
2246   \@@_rec_preamble_after_col:n
2247 }

2248 \cs_new:Npn \@@_r #1
2249 {
2250   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2251   \tl_gclear:N \g_@@_pre_cell_tl
2252   \tl_gput_right:Nn \g_@@_array_preamble_tl
2253     {
2254       > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2255       r
2256       < \@@_cell_end:
2257     }
2258   \int_gincr:N \c@jCol
2259   \@@_rec_preamble_after_col:n
2260 }

```

For ! and @

```

2261 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2262 {
2263   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2264   \@@_rec_preamble:n
2265 }
2266 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2267 \cs_new:cpn { @@ _ | } #1
2268 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2269   \int_incr:N \l_tmpa_int
2270   \@@_make_preamble_i_i:n
2271 }

2272 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2273 {
2274   \str_if_eq:nnTF { #1 } |
2275     { \use:c { @@ _ | } | }
2276     { \@@_make_preamble_i_ii:nn { } #1 }
2277 }

```

```

2278 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2279 {
2280   \str_if_eq:nnTF { #2 } [
2281     { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2282     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2283   ]
2284   \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2285   { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2286 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2287 {
2288   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2289   \tl_gput_right:Nx \g_@@_array_preamble_tl
2290   {

```

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

```

2291     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2292   }
2293   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2294   {
2295     \@@_vline:n
2296     {
2297       position = \int_eval:n { \c_jCol + 1 } ,
2298       multiplicity = \int_use:N \l_tmpa_int ,
2299       total-width = \dim_use:N \l_@@_rule_width_dim ,
2300       #2
2301     }

```

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.

```

2302   }
2303   \int_zero:N \l_tmpa_int
2304   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2305   \@@_rec_preamble:n #1
2306 }

2307 \cs_new:cpn { @@_ > } #1 #2
2308 {
2309   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2310   \@@_rec_preamble:n
2311 }

2312 \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.

```

2313 \keys_define:nn { WithArrows / p-column }
2314 {
2315   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2316   r .value_forbidden:n = true ,
2317   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2318   c .value_forbidden:n = true ,
2319   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2320   l .value_forbidden:n = true ,
2321   R .code:n =
2322     \IfPackageLoadedTF { ragged2e }
2323     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2324     {
2325       \@@_error_or_warning:n { ragged2e~not~loaded }
2326       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2327     } ,
2328   R .value_forbidden:n = true ,
2329   L .code:n =

```

```

2330 \IfPackageLoadedTF { ragged2e }
2331 { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2332 {
2333   \@@_error_or_warning:n { ragged2e~not~loaded }
2334   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2335 } ,
2336 L .value_forbidden:n = true ,
2337 C .code:n =
2338 \IfPackageLoadedTF { ragged2e }
2339 { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2340 {
2341   \@@_error_or_warning:n { ragged2e~not~loaded }
2342   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2343 } ,
2344 C .value_forbidden:n = true ,
2345 S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2346 S .value_forbidden:n = true ,
2347 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2348 p .value_forbidden:n = true ,
2349 t .meta:n = p ,
2350 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2351 m .value_forbidden:n = true ,
2352 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2353 b .value_forbidden:n = true ,
2354 }

```

For p, b and m.

```

2355 \cs_new:Npn \@@_p #1
2356 {
2357   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2358   \@@_make_preamble_ii_i:n
2359 }
2360 \cs_set_eq:NN \@@_b \@@_p
2361 \cs_set_eq:NN \@@_m \@@_p
2362 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2363 {
2364   \str_if_eq:nnTF { #1 } { [ ]
2365     { \@@_make_preamble_ii_ii:w [ ]
2366       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2367     }
2368   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2369     { \@@_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.

```

2370 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2371 {

```

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).

```

2372   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2373   % \tl_set:Nn \l_tmpa_tl { #1 }
2374   \@@_keys_p_column:n { #1 }
2375   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2376 }
2377 \cs_new_protected:Npn \@@_keys_p_column:n #1
2378 { \keys_set_known:nnN { WithArrows / 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.

```

2379 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2380 {
2381   \use:e
2382   {
2383     \@@_make_preamble_ii_v:nnnnnnnn
2384     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2385     { \dim_eval:n { #1 } }
2386     {

```

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.

```

2387       \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2388       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2389       {
2390         \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2391         { \str_lowercase:V \l_@@_hpos_col_str }
2392       }
2393       \str_case:on \l_@@_hpos_col_str
2394       {
2395         c { \exp_not:N \centering }
2396         l { \exp_not:N \raggedright }
2397         r { \exp_not:N \raggedleft }
2398         C { \exp_not:N \Centering }
2399         L { \exp_not:N \RaggedRight }
2400         R { \exp_not:N \RaggedLeft }
2401       }
2402       #3
2403     }
2404     { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2405     { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2406     { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2407     { #2 }
2408     {
2409       \str_case:onF \l_@@_hpos_col_str
2410       {
2411         { j } { c }
2412         { si } { c }
2413       }

```

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

```

2414       { \str_lowercase:V \l_@@_hpos_col_str }
2415     }
2416   }

```

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

```

2417   \int_gincr:N \c@jCol
2418   \@@_rec_preamble_after_col:n
2419 }

```

#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*.

```

2420 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2421 {
2422   \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2423     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2424     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2425   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2426   \tl_gclear:N \g_@@_pre_cell_tl
2427   \tl_gput_right:Nn \g_@@_array_preamble_tl
2428     {
2429       > {

```

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.

```

2430       \dim_set:Nn \l_@@_col_width_dim { #2 }
2431       \@@_cell_begin:w

```

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

```

2432       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2433       \everypar
2434       {
2435         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2436         \everypar { }
2437       }

```

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

```

2438       #3

```

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

```

2439       \g_@@_row_style_tl
2440       \arraybackslash
2441       #5
2442     }
2443     #8
2444     < {
2445       #6

```

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

```

2446       \@finalstrut \@arstrutbox
2447       \use:c { end #7 }

```

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

```

2448       #4
2449       \@@_cell_end:
2450     }
2451   }
2452 }

```

```

2453 \str_new:N \c_@@_ignorespaces_str
2454 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2455 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }

```

In order to test whether a cell is empty, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstances, for example when `\collectcell` of `collcell` 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...

```

2456 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2457 \cs_new_protected:Npn \@@_test_if_empty_i:

```

```

2458 {
2459   \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2460   \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2461   { \@@_test_if_empty:w }
2462 }
2463 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2464 {
2465   \peek_meaning:NT \unskip
2466   {
2467     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2468     {
2469       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2470       \skip_horizontal:N \l_@@_col_width_dim
2471     }
2472   }
2473 }
2474 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2475 {
2476   \peek_meaning:NT \__siunitx_table_skip:n
2477   {
2478     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2479     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2480   }
2481 }

```

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

```

2482 \cs_new_protected:Npn \@@_center_cell_box:
2483 {

```

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

```

2484   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2485   {
2486     \int_compare:nNnT
2487     { \box_ht:N \l_@@_cell_box }
2488     >

```

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).

```

2489     { \box_ht:N \strutbox }
2490     {
2491       \hbox_set:Nn \l_@@_cell_box
2492       {
2493         \box_move_down:nn
2494         {
2495           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2496             + \baselineskip ) / 2
2497         }
2498         { \box_use:N \l_@@_cell_box }
2499       }
2500     }
2501   }
2502 }

```

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

```

2503 \cs_new:Npn \@@_V #1 #2
2504 {

```

```

2505 \str_if_eq:nnTF { #2 } { [ ] }
2506 { \@@_make_preamble_V_i:w [ ] }
2507 { \@@_make_preamble_V_i:w [ ] { #2 } }
2508 }
2509 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2510 { \@@_make_preamble_V_ii:nn { #1 } }
2511 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2512 {
2513 \str_set:Nn \l_@@_vpos_col_str { p }
2514 \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2515 \@@_keys_p_column:n { #1 }
2516 \IfPackageLoadedTF { varwidth }
2517 { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2518 {
2519 \@@_error_or_warning:n { varwidth-not-loaded }
2520 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2521 }
2522 }

```

For w and W

```

2523 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2524 \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.

```

2525 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2526 {
2527 \str_if_eq:nnTF { #3 } { s }
2528 { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2529 { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2530 }

```

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.

```

2531 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2532 {
2533 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2534 \tl_gclear:N \g_@@_pre_cell_tl
2535 \tl_gput_right:Nn \g_@@_array_preamble_tl
2536 {
2537 > {
2538 \dim_set:Nn \l_@@_col_width_dim { #2 }
2539 \@@_cell_begin:w
2540 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2541 }
2542 c
2543 < {
2544 \@@_cell_end_for_w_s:
2545 #1
2546 \@@_adjust_size_box:
2547 \box_use_drop:N \l_@@_cell_box
2548 }
2549 }
2550 \int_gincr:N \c@jCol
2551 \@@_rec_preamble_after_col:n
2552 }

```

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

```

2553 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4

```

```

2554 {
2555   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2556   \tl_gclear:N \g_@@_pre_cell_tl
2557   \tl_gput_right:Nn \g_@@_array_preamble_tl
2558   {
2559     > {

```

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.

```

2560       \dim_set:Nn \l_@@_col_width_dim { #4 }
2561       \hbox_set:Nw \l_@@_cell_box
2562       \@@_cell_begin:w
2563       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2564     }
2565     c
2566     < {
2567       \@@_cell_end:
2568       \hbox_set_end:
2569       #1
2570       \@@_adjust_size_box:
2571       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2572     }
2573   }

```

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

```

2574   \int_gincr:N \c@jCol
2575   \@@_rec_preamble_after_col:n
2576 }

2577 \cs_new_protected:Npn \@@_special_W:
2578 {
2579   \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2580   { \@@_warning:n { W~warning } }
2581 }

```

For `S` (of `siunitx`).

```

2582 \cs_new:Npn \@@_S #1 #2
2583 {
2584   \str_if_eq:nnTF { #2 } { [ ] }
2585   { \@@_make_preamble_S:w [ ] }
2586   { \@@_make_preamble_S:w [ ] { #2 } }
2587 }

2588 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2589 { \@@_make_preamble_S:i:n { #1 } }

2590 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2591 {
2592   \IfPackageLoadedTF { siunitx }
2593   {
2594     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2595     \tl_gclear:N \g_@@_pre_cell_tl
2596     \tl_gput_right:Nn \g_@@_array_preamble_tl
2597     {
2598       > {
2599         \@@_cell_begin:w
2600         \keys_set:nn { siunitx } { #1 }
2601         \siunitx_cell_begin:w
2602       }
2603       c
2604       < { \siunitx_cell_end: \@@_cell_end: }
2605     }

```

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

```

2606         \int_gincr:N \c@jCol
2607         \@@_rec_preamble_after_col:n
2608     }
2609     { \@@_fatal:n { siunitx~not~loaded } }
2610 }

```

For (, [and \{.

```

2611 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2612 {
2613     \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.

```

2614     \int_if_zero:nTF \c@jCol
2615     {
2616         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2617         {

```

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

```

2618             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2619             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2620             \@@_rec_preamble:n #2
2621         }
2622     {
2623         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2624         \@@_make_preamble_iv:nn { #1 } { #2 }
2625     }
2626 }
2627 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2628 }
2629 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2630 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2631 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2632 {
2633     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2634     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2635     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2636     {
2637         \@@_error:nn { delimiter~after~opening } { #2 }
2638         \@@_rec_preamble:n
2639     }
2640     { \@@_rec_preamble:n #2 }
2641 }

```

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

```

2642 \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}).

```

2643 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2644 {
2645     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2646     \tl_if_in:nnTF { ) ] \} } { #2 }
2647     { \@@_make_preamble_v:nnn #1 #2 }
2648     {
2649         \tl_if_eq:nnTF { \stop } { #2 }
2650         {
2651             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2652             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2653             {

```

```

2654         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2655         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2656         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2657         \@@_rec_preamble:n #2
2658     }
2659 }
2660 {
2661     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2662     { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2663     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2664     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2665     \@@_rec_preamble:n #2
2666 }
2667 }
2668 }
2669 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ] }
2670 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2671 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2672 {
2673     \tl_if_eq:nnTF { \stop } { #3 }
2674     {
2675         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2676         {
2677             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2678             \tl_gput_right:Nx \g_@@_pre_code_after_tl
2679             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2680             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2681         }
2682         {
2683             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2684             \tl_gput_right:Nx \g_@@_pre_code_after_tl
2685             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2686             \@@_error:nn { double~closing~delimiter } { #2 }
2687         }
2688     }
2689     {
2690         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2691         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2692         \@@_error:nn { double~closing~delimiter } { #2 }
2693         \@@_rec_preamble:n #3
2694     }
2695 }
2696 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2697 { \use:c { @@ _ \token_to_str:N ) } }

```

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

```

2698 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2699 {
2700     \str_if_eq:nnTF { #1 } { < }
2701     \@@_rec_preamble_after_col_i:n
2702     {
2703         \str_if_eq:nnTF { #1 } { @ }
2704         \@@_rec_preamble_after_col_ii:n
2705         {
2706             \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2707             {
2708                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2709                 { ! { \skip_horizontal:N \arrayrulewidth } }
2710             }

```

```

2711         {
2712             \exp_args:NNe
2713             \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2714             {
2715                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2716                 { ! { \skip_horizontal:N \arrayrulewidth } }
2717             }
2718         }
2719         \@@_rec_preamble:n { #1 }
2720     }
2721 }
2722 }
2723 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2724 {
2725     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2726     \@@_rec_preamble_after_col:n
2727 }

```

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

```

2728 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2729 {
2730     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2731     {
2732         \tl_gput_right:Nn \g_@@_array_preamble_tl
2733         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2734     }
2735     {
2736         \exp_args:NNe
2737         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2738         {
2739             \tl_gput_right:Nn \g_@@_array_preamble_tl
2740             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2741         }
2742         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2743     }
2744     \@@_rec_preamble:n
2745 }

```

```

2746 \cs_new:cpn { @@ _ * } #1 #2 #3
2747 {
2748     \tl_clear:N \l_tmpa_tl
2749     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2750     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2751 }

```

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.

```

2752 \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.

```

2753 \cs_new:Npn \@@_X #1 #2
2754 {
2755     \str_if_eq:nnTF { #2 } { [ ]
2756         { \@@_make_preamble_X:w [ ] }
2757         { \@@_make_preamble_X:w [ ] #2 }
2758     }
2759 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2760 { \@@_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 { `WithArrows` / `p-column` } but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter `\l_@@_weight_int`).

```
2761 \keys_define:nn { WithArrows / X-column }
2762 { 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.

```
2763 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2764 {
```

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).

```
2765 \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).

```
2766 \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`.

```
2767 \int_zero_new:N \l_@@_weight_int
2768 \int_set_eq:NN \l_@@_weight_int \c_one_int
2769 \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2770 \keys_set:no { WithArrows / X-column } \l_tmpa_tl
2771 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2772 {
2773   \@@_error_or_warning:n { negative-weight }
2774   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2775 }
2776 \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).

```
2777 \bool_if:NTF \l_@@_X_columns_aux_bool
2778 {
2779   \exp_args:Nne
2780   \@@_make_preamble_ii_iv:nnn
2781   { \l_@@_weight_int \l_@@_X_columns_dim }
2782   { minipage }
2783   { \@@_no_update_width: }
2784 }
2785 {
2786   \tl_gput_right:Nn \g_@@_array_preamble_tl
2787   {
2788     > {
2789       \@@_cell_begin:w
2790       \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`.

```
2791 \NotEmpty
```

The following code will nullify the box of the cell.

```
2792 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2793 { \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`.

```

2794         \begin { minipage } { 5 cm } \arraybackslash
2795     }
2796     c
2797     < {
2798         \end { minipage }
2799         \@@_cell_end:
2800     }
2801 }
2802 \int_gincr:N \c@jCol
2803 \@@_rec_preamble_after_col:n
2804 }
2805 }

2806 \cs_new_protected:Npn \@@_no_update_width:
2807 {
2808     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2809     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2810 }

```

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

```

2811 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2812 {
2813     \seq_gput_right:Nx \g_@@_cols_vlism_seq
2814     { \int_eval:n { \c@jCol + 1 } }
2815     \tl_gput_right:Nx \g_@@_array_preamble_tl
2816     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2817     \@@_rec_preamble:n
2818 }

```

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.

```

2819 \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).

```

2820 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2821 { \@@_fatal:n { Preamble-forgotten } }
2822 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2823 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2824 \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).

```

2825 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2826 {

```

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`.

```

2827     \multispan { #1 }
2828     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2829     \begingroup
2830     \cs_set:Npn \@addamp
2831     { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```
2832 \tl_gclear:N \g_@@_preamble_tl
2833 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2834 \exp_args:No \@mkpream \g_@@_preamble_tl
2835 \@addtopreamble \empty
2836 \endgroup
```

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

```
2837 \int_compare:nNt { #1 } > \c_one_int
2838 {
2839   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2840   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2841   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2842   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2843   {
2844     {
2845       \int_if_zero:nTF \c@jCol
2846       { \int_eval:n { \c@iRow + 1 } }
2847       { \int_use:N \c@iRow }
2848     }
2849     { \int_eval:n { \c@jCol + 1 } }
2850     {
2851       \int_if_zero:nTF \c@jCol
2852       { \int_eval:n { \c@iRow + 1 } }
2853       { \int_use:N \c@iRow }
2854     }
2855     { \int_eval:n { \c@jCol + #1 } }
2856     { } % for the name of the block
2857   }
2858 }
```

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

```
2859 \cs_set:Npn \@sharp { #3 }
2860 \@arstrut
2861 \@preamble
2862 \null
```

We add some lines.

```
2863 \int_gadd:Nn \c@jCol { #1 - 1 }
2864 \int_compare:nNt \c@jCol > \g_@@_col_total_int
2865 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2866 \ignorespaces
2867 }
```

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`.

```
2868 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2869 {
2870   \str_case:nnF { #1 }
2871   {
2872     c { \@@_make_m_preamble_i:n #1 }
2873     l { \@@_make_m_preamble_i:n #1 }
2874     r { \@@_make_m_preamble_i:n #1 }
2875     > { \@@_make_m_preamble_ii:nn #1 }
2876     ! { \@@_make_m_preamble_ii:nn #1 }
2877     @ { \@@_make_m_preamble_ii:nn #1 }
2878     | { \@@_make_m_preamble_iii:n #1 }
2879     p { \@@_make_m_preamble_iv:nnn t #1 }
2880     m { \@@_make_m_preamble_iv:nnn c #1 }
```

```

2881     b { \@@_make_m_preamble_iv:nnn b #1 }
2882     w { \@@_make_m_preamble_v:nnnn { } #1 }
2883     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2884     \q_stop { }
2885   }
2886   {
2887     \cs_if_exist:cTF { NC @ find @ #1 }
2888     {
2889       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2890       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2891     }
2892     {
2893       \tl_if_eq:nnT { #1 } { S }
2894       { \@@_fatal:n { unknown~column~type~S } }
2895       { \@@_fatal:nn { unknown~column~type } { #1 } }
2896     }
2897   }
2898 }

```

For c, l and r

```

2899 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2900 {
2901   \tl_gput_right:Nn \g_@@_preamble_tl
2902   {
2903     > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2904     #1
2905     < \@@_cell_end:
2906   }

```

We test for the presence of a <.

```

2907   \@@_make_m_preamble_x:n
2908 }

```

For >, ! and @

```

2909 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2910 {
2911   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2912   \@@_make_m_preamble:n
2913 }

```

For |

```

2914 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2915 {
2916   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2917   \@@_make_m_preamble:n
2918 }

```

For p, m and b

```

2919 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2920 {
2921   \tl_gput_right:Nn \g_@@_preamble_tl
2922   {
2923     > {
2924       \@@_cell_begin:w
2925       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2926       \mode_leave_vertical:
2927       \arraybackslash
2928       \vrule height \box_ht:N \@@_arstrutbox depth 0 pt width 0 pt
2929     }
2930     c
2931     < {
2932       \vrule height 0 pt depth \box_dp:N \@@_arstrutbox width 0 pt
2933       \end { minipage }

```

```

2934         \@@_cell_end:
2935     }
2936 }

```

We test for the presence of a <.

```

2937     \@@_make_m_preamble_x:n
2938 }

```

For w and W

```

2939 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2940 {
2941     \tl_gput_right:Nn \g_@@_preamble_tl
2942     {
2943         > {
2944             \dim_set:Nn \l_@@_col_width_dim { #4 }
2945             \hbox_set:Nw \l_@@_cell_box
2946             \@@_cell_begin:w
2947             \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2948         }
2949         c
2950         < {
2951             \@@_cell_end:
2952             \hbox_set_end:
2953             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2954             #1
2955             \@@_adjust_size_box:
2956             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2957         }
2958     }

```

We test for the presence of a <.

```

2959     \@@_make_m_preamble_x:n
2960 }

```

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

```

2961 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2962 {
2963     \str_if_eq:nnTF { #1 } { < }
2964     \@@_make_m_preamble_ix:n
2965     { \@@_make_m_preamble:n { #1 } }
2966 }
2967 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2968 {
2969     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2970     \@@_make_m_preamble_x:n
2971 }

```

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`).

```

2972 \cs_new_protected:Npn \@@_put_box_in_flow:
2973 {
2974     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2975     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2976     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2977     { \box_use_drop:N \l_tmpa_box }
2978     \@@_put_box_in_flow_i:
2979 }

```

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).

```

2980 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2981 {
2982   \pgfpicture
2983   \@@_qpoint:n { row - 1 }
2984   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2985   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2986   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2987   \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).

```

2988   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2989   {
2990     \int_set:Nn \l_tmpa_int
2991     {
2992       \str_range:Nnn
2993       \l_@@_baseline_tl
2994       6
2995       { \tl_count:o \l_@@_baseline_tl }
2996     }
2997     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2998   }
2999   {
3000     \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3001     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3002     {
3003       \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3004       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3005       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3006     }
3007     \bool_lazy_or:nnT
3008     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3009     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3010     {
3011       \@@_error:n { bad-value-for-baseline }
3012       \int_set_eq:NN \l_tmpa_int \c_one_int
3013     }
3014     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3015     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3016   }
3017   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3018   \endpgfpicture
3019   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3020   \box_use_drop:N \l_tmpa_box
3021 }

```

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).

```

3022 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3023 {

```

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.

```

3024   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3025   {
3026     \int_compare:nNnT \c@jCol > \c_one_int
3027     {
3028       \box_set_wd:Nn \l_@@_the_array_box

```

```

3029         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3030     }
3031 }

```

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

```

3032 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3033 \bool_if:NT \l_@@_caption_above_bool
3034 {
3035     \tl_if_empty:NF \l_@@_caption_tl
3036     {
3037         \bool_set_false:N \g_@@_caption_finished_bool
3038         \int_gzero:N \c@tabularnote
3039         \@@_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).

```

3040         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3041         {
3042             \tl_build_gput_right:Nx \g_@@_aux_tl
3043             {
3044                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3045                 { \int_use:N \g_@@_notes_caption_int }
3046             }
3047             \int_gzero:N \g_@@_notes_caption_int
3048         }
3049     }
3050 }

```

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

```

3051 \hbox
3052 {
3053     \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.

```

3054 \@@_create_extra_nodes:
3055 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3056 }

```

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

```

3057 \bool_lazy_any:nT
3058 {
3059     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3060     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3061     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3062 }
3063 \@@_insert_tabularnotes:
3064 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3065 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3066 \end { minipage }
3067 }

```

```

3068 \cs_new_protected:Npn \@@_insert_caption:
3069 {
3070     \tl_if_empty:NF \l_@@_caption_tl
3071     {

```

```

3072     \cs_if_exist:NTF \@capytype
3073     { \@@_insert_caption_i: }
3074     { \@@_error:n { caption~outside~float } }
3075   }
3076 }

```

```

3077 \cs_new_protected:Npn \@@_insert_caption_i:
3078 {
3079   \group_begin:

```

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

```

3080   \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.

```

3081   \IfPackageLoadedTF { floatrow }
3082   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3083   { }
3084   \tl_if_empty:NTF \l_@@_short_caption_tl
3085   { \caption }
3086   { \caption [ \l_@@_short_caption_tl ] }
3087   { \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.

```

3088   \bool_if:NF \g_@@_caption_finished_bool
3089   {
3090     \bool_gset_true:N \g_@@_caption_finished_bool
3091     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3092     \int_gzero:N \c@tabularnote
3093   }
3094   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3095   \group_end:
3096 }

```

```

3097 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3098 {
3099   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3100   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3101 }
3102 \cs_new_protected:Npn \@@_insert_tabularnotes:
3103 {
3104   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3105   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3106   \skip_vertical:N 0.65ex

```

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

```

3107   \group_begin:
3108   \l_@@_notes_code_before_tl
3109   \tl_if_empty:NF \g_@@_tabularnote_tl
3110   {
3111     \g_@@_tabularnote_tl \par
3112     \tl_gclear:N \g_@@_tabularnote_tl
3113   }

```

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.

```

3114   \int_compare:nNnT \c@tabularnote > \c_zero_int
3115   {

```

```

3116 \bool_if:NTF \l_@@_notes_para_bool
3117 {
3118   \begin { tabularnotes* }
3119     \seq_map_inline:Nn \g_@@_notes_seq
3120       { \@@_one_tabularnote:nn ##1 }
3121     \strut
3122   \end { tabularnotes* }

```

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

```

3123 \par
3124 }
3125 {
3126   \tabularnotes
3127     \seq_map_inline:Nn \g_@@_notes_seq
3128       { \@@_one_tabularnote:nn ##1 }
3129     \strut
3130   \endtabularnotes
3131 }
3132 }
3133 \unskip
3134 \group_end:
3135 \bool_if:NT \l_@@_notes_bottomrule_bool
3136 {
3137   \IfPackageLoadedTF { booktabs }
3138   {

```

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

```

3139 \skip_vertical:N \aboverulesep

```

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

```

3140 { \CT@arc@ \hrule height \heavyrulewidth }
3141 }
3142 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3143 }
3144 \l_@@_notes_code_after_tl
3145 \seq_gclear:N \g_@@_notes_seq
3146 \seq_gclear:N \g_@@_notes_in_caption_seq
3147 \int_gzero:N \c@tabularnote
3148 }

```

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.

```

3149 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3150 {
3151   \tl_if_novalue:nTF { #1 }
3152   { \item }
3153   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3154 }

```

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`.

```

3155 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3156 {
3157   \pgfpicture
3158     \@@_qpoint:n { row - 1 }
3159     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3160     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3161     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3162   \endpgfpicture
3163   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth

```



```

3164 \int_if_zero:nT \l_@@_first_row_int
3165 {
3166     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3167     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3168 }
3169 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3170 }

```

Now, the general case.

```

3171 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3172 {

```

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

```

3173 \tl_if_eq:NnT \l_@@_baseline_tl { t }
3174 { \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`.

```

3175 \pgfpicture
3176 \@@_qpoint:n { row - 1 }
3177 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3178 \str_if_in:NnTF \l_@@_baseline_tl { line- }
3179 {
3180     \int_set:Nn \l_tmpa_int
3181     {
3182         \str_range:Nnn
3183         \l_@@_baseline_tl
3184         6
3185         { \tl_count:o \l_@@_baseline_tl }
3186     }
3187     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3188 }
3189 {
3190     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3191     \bool_lazy_or:nnT
3192     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3193     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3194     {
3195         \@@_error:n { bad~value~for~baseline }
3196         \int_set:Nn \l_tmpa_int 1
3197     }
3198     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3199 }
3200 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3201 \endpgfpicture
3202 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3203 \int_if_zero:nT \l_@@_first_row_int
3204 {
3205     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3206     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3207 }
3208 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3209 }

```

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.

```

3210 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3211 {

```

We will compute the real width of both delimiters used.

```

3212 \dim_zero_new:N \l_@@_real_left_delim_dim
3213 \dim_zero_new:N \l_@@_real_right_delim_dim
3214 \hbox_set:Nn \l_tmpb_box

```

```

3215 {
3216   \c_math_toggle_token
3217   \left #1
3218   \vcenter
3219   {
3220     \vbox_to_ht:nn
3221     { \box_ht_plus_dp:N \l_tmpa_box }
3222     { }
3223   }
3224   \right .
3225   \c_math_toggle_token
3226 }
3227 \dim_set:Nn \l_@@_real_left_delim_dim
3228 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3229 \hbox_set:Nn \l_tmpb_box
3230 {
3231   \c_math_toggle_token
3232   \left .
3233   \vbox_to_ht:nn
3234   { \box_ht_plus_dp:N \l_tmpa_box }
3235   { }
3236   \right #2
3237   \c_math_toggle_token
3238 }
3239 \dim_set:Nn \l_@@_real_right_delim_dim
3240 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3241   \skip_horizontal:N \l_@@_left_delim_dim
3242   \skip_horizontal:N -\l_@@_real_left_delim_dim
3243   \@@_put_box_in_flow:
3244   \skip_horizontal:N \l_@@_right_delim_dim
3245   \skip_horizontal:N -\l_@@_real_right_delim_dim
3246 }

```

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).

```

3247 \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).

```

3248 {
3249   \peek_remove_spaces:n
3250   {
3251     \peek_meaning:NTF \end
3252     \@@_analyze_end:Nn
3253     {
3254       \@@_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`).

```

3255     \exp_args:No \@@_array: \g_@@_array_preamble_tl
3256   }
3257 }
3258 }
3259 {
3260   \@@_create_col_nodes:
3261   \endarray
3262 }

```

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

```
3263 \NewDocumentEnvironment { @@-light-syntax } { b }
3264 {
```

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`.

```
3265 \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3266 \tl_map_inline:nn { #1 }
3267 {
3268   \str_if_eq:nnT { ##1 } { & }
3269   { \@@_fatal:n { ampersand-in~light-syntax } }
3270   \str_if_eq:nnT { ##1 } { \ }
3271   { \@@_fatal:n { double-backslash-in~light-syntax } }
3272 }
```

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`.

```
3273 \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3274 }
```

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.

```
3275 {
3276   \@@_create_col_nodes:
3277   \endarray
3278 }
3279 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3280 {
3281   \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).

```
3282 \seq_clear_new:N \l_@@_rows_seq
```

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

```
3283 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3284 \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3285 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3286 \tl_if_empty:NF \l_tmpa_tl
3287 { \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.

```
3288 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3289 { \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`.

```
3290 \tl_build_begin:N \l_@@_new_body_tl
3291 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3292 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3293 \@@_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).

```

3294 \seq_map_inline:Nn \l_@@_rows_seq
3295 {
3296   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3297   \@@_line_with_light_syntax:n { #1 }
3298 }
3299 \tl_build_end:N \l_@@_new_body_tl
3300 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3301 {
3302   \int_set:Nn \l_@@_last_col_int
3303     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3304 }

```

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.

```

3305 \@@_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`).

```

3306 \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3307 }
3308 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3309 {
3310   \seq_clear_new:N \l_@@_cells_seq
3311   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3312   \int_set:Nn \l_@@_nb_cols_int
3313     {
3314       \int_max:nn
3315         \l_@@_nb_cols_int
3316         { \seq_count:N \l_@@_cells_seq }
3317     }
3318   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3319   \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3320   \seq_map_inline:Nn \l_@@_cells_seq
3321     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3322 }
3323 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }

```

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`.

```

3324 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3325 {
3326   \str_if_eq:onT \g_@@_name_env_str { #2 }
3327   { \@@_fatal:n { empty-environment } }

```

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

```

3328   \end { #2 }
3329 }

```

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).

```

3330 \cs_new:Npn \@@_create_col_nodes:
3331 {
3332   \crrc
3333   \int_if_zero:nT \l_@@_first_col_int
3334   {
3335     \omit

```

```

3336 \hbox_overlap_left:n
3337 {
3338   \bool_if:NT \l_@@_code_before_bool
3339   { \pgfsys@markposition { \@@_env: - col - 0 } }
3340   \pgfpicture
3341   \pgfrememberpicturepositiononpagetrue
3342   \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3343   \str_if_empty:NF \l_@@_name_str
3344   { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3345   \endpgfpicture
3346   \skip_horizontal:N 2\col@sep
3347   \skip_horizontal:N \g_@@_width_first_col_dim
3348 }
3349 &
3350 }
3351 \omit

```

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

```

3352 \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`).

```

3353 \int_if_zero:nTF \l_@@_first_col_int
3354 {
3355   \bool_if:NT \l_@@_code_before_bool
3356   {
3357     \hbox
3358     {
3359       \skip_horizontal:N -0.5\arrayrulewidth
3360       \pgfsys@markposition { \@@_env: - col - 1 }
3361       \skip_horizontal:N 0.5\arrayrulewidth
3362     }
3363   }
3364   \pgfpicture
3365   \pgfrememberpicturepositiononpagetrue
3366   \pgfcoordinate { \@@_env: - col - 1 }
3367   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3368   \str_if_empty:NF \l_@@_name_str
3369   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3370   \endpgfpicture
3371 }
3372 {
3373   \bool_if:NT \l_@@_code_before_bool
3374   {
3375     \hbox
3376     {
3377       \skip_horizontal:N 0.5\arrayrulewidth
3378       \pgfsys@markposition { \@@_env: - col - 1 }
3379       \skip_horizontal:N -0.5\arrayrulewidth
3380     }
3381   }
3382   \pgfpicture
3383   \pgfrememberpicturepositiononpagetrue
3384   \pgfcoordinate { \@@_env: - col - 1 }
3385   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3386   \str_if_empty:NF \l_@@_name_str
3387   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3388   \endpgfpicture
3389 }

```

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.

```

3390 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3391 \bool_if:NF \l_@@_auto_columns_width_bool
3392 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3393 {
3394   \bool_lazy_and:nnTF
3395     \l_@@_auto_columns_width_bool
3396     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3397     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3398     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3399   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3400 }
3401 \skip_horizontal:N \g_tmpa_skip
3402 \hbox
3403 {
3404   \bool_if:NT \l_@@_code_before_bool
3405   {
3406     \hbox
3407     {
3408       \skip_horizontal:N -0.5\arrayrulewidth
3409       \pgfsys@markposition { \@@_env: - col - 2 }
3410       \skip_horizontal:N 0.5\arrayrulewidth
3411     }
3412   }
3413   \pgfpicture
3414   \pgfrememberpicturerepositiononpagetrue
3415   \pgfcoordinate { \@@_env: - col - 2 }
3416   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3417   \str_if_empty:NF \l_@@_name_str
3418   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3419   \endpgfpicture
3420 }

```

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.

```

3421 \int_gset_eq:NN \g_tmpa_int \c_one_int
3422 \bool_if:NTF \g_@@_last_col_found_bool
3423 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3424 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3425 {
3426   &
3427   \omit
3428   \int_gincr:N \g_tmpa_int

```

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

```

3429 \skip_horizontal:N \g_tmpa_skip
3430 \bool_if:NT \l_@@_code_before_bool
3431 {
3432   \hbox
3433   {
3434     \skip_horizontal:N -0.5\arrayrulewidth
3435     \pgfsys@markposition
3436     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3437     \skip_horizontal:N 0.5\arrayrulewidth
3438   }
3439 }

```

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

```

3440 \pgfpicture
3441 \pgfrememberpicturerepositiononpagetrue
3442 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3443 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3444 \str_if_empty:NF \l_@@_name_str

```

```

3445         {
3446             \pgfnodealias
3447             { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3448             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3449         }
3450     \endpgfpicture
3451 }

3452 &
3453 \omit

```

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

```

3454 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
3455 { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3456 \skip_horizontal:N \g_tmpa_skip
3457 \int_gincr:N \g_tmpa_int
3458 \bool_lazy_any:nT
3459 {
3460     \g_@@_delims_bool
3461     \l_@@_tabular_bool
3462     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3463     \l_@@_exterior_arraycolsep_bool
3464     \l_@@_bar_at_end_of_pream_bool
3465 }
3466 { \skip_horizontal:N -\col@sep }
3467 \bool_if:NT \l_@@_code_before_bool
3468 {
3469     \hbox
3470     {
3471         \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.

```

3472 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3473 { \skip_horizontal:N -\arraycolsep }
3474 \pgfsys@markposition
3475 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3476 \skip_horizontal:N 0.5\arrayrulewidth
3477 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3478 { \skip_horizontal:N \arraycolsep }
3479 }
3480 }
3481 \pgfpicture
3482 \pgfrememberpicturepositiononpagetrue
3483 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3484 {
3485     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3486     {
3487         \pgfpoint
3488         { - 0.5 \arrayrulewidth - \arraycolsep }
3489         \c_zero_dim
3490     }
3491     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3492 }
3493 \str_if_empty:NF \l_@@_name_str
3494 {
3495     \pgfnodealias
3496     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3497     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3498 }
3499 \endpgfpicture

```

```

3500 \bool_if:NT \g_@@_last_col_found_bool
3501 {
3502   \hbox_overlap_right:n
3503   {
3504     \skip_horizontal:N \g_@@_width_last_col_dim
3505     \skip_horizontal:N \col@sep % added 2023-11-05
3506     \bool_if:NT \l_@@_code_before_bool
3507     {
3508       \pgfsys@markposition
3509       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3510     }
3511     \pgfpicture
3512     \pgfrememberpicturerepositiononpagetrue
3513     \pgfcoordinate
3514     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3515     \pgfpointorigin
3516     \str_if_empty:NF \l_@@_name_str
3517     {
3518       \pgfnodealias
3519       {
3520         \l_@@_name_str - col
3521         - \int_eval:n { \g_@@_col_total_int + 1 }
3522       }
3523       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3524     }
3525     \endpgfpicture
3526   }
3527 }
3528 \cr
3529 }

```

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

```

3530 \tl_const:Nn \c_@@_preamble_first_col_tl
3531 {
3532   >
3533   {

```

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).

```

3534 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3535 \bool_gset_true:N \g_@@_after_col_zero_bool
3536 \@@_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.

```

3537 \hbox_set:Nw \l_@@_cell_box
3538 \@@_math_toggle:
3539 \@@_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”.

```

3540 \int_compare:nNnT \c@iRow > \c_zero_int
3541 {
3542   \bool_lazy_or:nnT
3543   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3544   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3545   {
3546     \l_@@_code_for_first_col_tl
3547     \xglobal \colorlet { nicematrix-first-col } { . }
3548   }
3549 }
3550 }

```


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`.

```

3551     l
3552     <
3553     {
3554         \@@_math_toggle:
3555         \hbox_set_end:
3556         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3557         \@@_adjust_size_box:
3558         \@@_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.

```

3559         \dim_gset:Nn \g_@@_width_first_col_dim
3560         { \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.

```

3561         \hbox_overlap_left:n
3562         {
3563             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3564             \@@_node_for_cell:
3565             { \box_use_drop:N \l_@@_cell_box }
3566             \skip_horizontal:N \l_@@_left_delim_dim
3567             \skip_horizontal:N \l_@@_left_margin_dim
3568             \skip_horizontal:N \l_@@_extra_left_margin_dim
3569         }
3570         \bool_gset_false:N \g_@@_empty_cell_bool
3571         \skip_horizontal:N -2\col@sep
3572     }
3573 }

```

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

```

3574 \tl_const:Nn \c_@@_preamble_last_col_tl
3575 {
3576     >
3577     {
3578         \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).

```

3579         \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.

```

3580         \bool_gset_true:N \g_@@_last_col_found_bool
3581         \int_gincr:N \c@jCol
3582         \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.

```

3583         \hbox_set:Nw \l_@@_cell_box
3584         \@@_math_toggle:
3585         \@@_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”.

```

3586         \int_compare:nNnT \c@iRow > \c_zero_int
3587         {
3588             \bool_lazy_or:nnT
3589             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3590             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3591             {
3592                 \l_@@_code_for_last_col_tl
3593                 \xglobal \colorlet { nicematrix-last-col } { . }
3594             }
3595         }
3596     }

```

```

3597     1
3598     <
3599     {
3600         \@@_math_toggle:
3601         \hbox_set_end:
3602         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3603         \@@_adjust_size_box:
3604         \@@_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.

```

3605         \dim_gset:Nn \g_@@_width_last_col_dim
3606         { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3607         \skip_horizontal:N -2\col@sep

```

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

```

3608         \hbox_overlap_right:n
3609         {
3610             \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3611             {
3612                 \skip_horizontal:N \l_@@_right_delim_dim
3613                 \skip_horizontal:N \l_@@_right_margin_dim
3614                 \skip_horizontal:N \l_@@_extra_right_margin_dim
3615                 \@@_node_for_cell:
3616             }
3617         }
3618         \bool_gset_false:N \g_@@_empty_cell_bool
3619     }
3620 }

```

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

```

3621 \NewDocumentEnvironment { NiceArray } { }
3622 {
3623     \bool_gset_false:N \g_@@_delims_bool
3624     \str_if_empty:NT \g_@@_name_env_str
3625     { \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).

```

3626     \NiceArrayWithDelims . .
3627 }
3628 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3629 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3630 {
3631     \NewDocumentEnvironment { #1 NiceArray } { }
3632     {
3633         \bool_gset_true:N \g_@@_delims_bool
3634         \str_if_empty:NT \g_@@_name_env_str
3635         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3636         \@@_test_if_math_mode:
3637         \NiceArrayWithDelims #2 #3
3638     }
3639     { \endNiceArrayWithDelims }
3640 }
3641 \@@_def_env:nnn p ( )
3642 \@@_def_env:nnn b [ ]
3643 \@@_def_env:nnn B \{ \}
3644 \@@_def_env:nnn v | |
3645 \@@_def_env:nnn V \l \r

```

14 The environment `{NiceMatrix}` and its variants

```

3646 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3647 {
3648   \bool_set_false:N \l_@@_preamble_bool
3649   \tl_clear:N \l_tmpa_tl
3650   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3651     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3652   \tl_put_right:Nn \l_tmpa_tl
3653     {
3654       *
3655       {
3656         \int_case:nnF \l_@@_last_col_int
3657           {
3658             { -2 } { \c@MaxMatrixCols }
3659             { -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).

```

3660       }
3661       { \int_eval:n { \l_@@_last_col_int - 1 } }
3662     }
3663     { #2 }
3664   }
3665   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3666   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3667 }
3668 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3669 \clist_map_inline:nn { p , b , B , v , V }
3670 {
3671   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3672   {
3673     \bool_gset_true:N \g_@@_delims_bool
3674     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3675     % added 2023/10/01
3676     \int_if_zero:nT \l_@@_last_col_int
3677       {
3678         \bool_set_true:N \l_@@_last_col_without_value_bool
3679         \int_set:Nn \l_@@_last_col_int { -1 }
3680       }
3681     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3682     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3683   }
3684   { \use:c { end #1 NiceArray } }
3685 }

```

We define also an environment `{NiceMatrix}`

```

3686 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3687 {
3688   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3689   % added 2023/10/01
3690   \int_if_zero:nT \l_@@_last_col_int
3691     {
3692       \bool_set_true:N \l_@@_last_col_without_value_bool
3693       \int_set:Nn \l_@@_last_col_int { -1 }
3694     }
3695   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3696   \bool_lazy_or:nnT
3697     { \clist_if_empty_p:N \l_@@_vlines_clist }
3698     { \l_@@_except_borders_bool }
3699     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3700   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3701 }

```

```
3702 { \endNiceArray }
```

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

```
3703 \cs_new_protected:Npn \@@_NotEmpty:
3704 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3705 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3706 {
```

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`.

```
3707   \dim_compare:nNtT \l_@@_width_dim = \c_zero_dim
3708   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3709   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3710   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3711   \tl_if_empty:NF \l_@@_short_caption_tl
3712   {
3713     \tl_if_empty:NT \l_@@_caption_tl
3714     {
3715       \@@_error_or_warning:n { short-caption-without-caption }
3716       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3717     }
3718   }
3719   \tl_if_empty:NF \l_@@_label_tl
3720   {
3721     \tl_if_empty:NT \l_@@_caption_tl
3722     { \@@_error_or_warning:n { label-without-caption } }
3723   }
3724   \NewDocumentEnvironment { TabularNote } { b }
3725   {
3726     \bool_if:NTF \l_@@_in_code_after_bool
3727     { \@@_error_or_warning:n { TabularNote~in-CodeAfter } }
3728     {
3729       \tl_if_empty:NF \g_@@_tabularnote_tl
3730       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3731       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3732     }
3733   }
3734   { }
3735   \@@_settings_for_tabular:
3736   \NiceArray { #2 }
3737 }
3738 { \endNiceArray }
3739 \cs_new_protected:Npn \@@_settings_for_tabular:
3740 {
3741   \bool_set_true:N \l_@@_tabular_bool
3742   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3743   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3744   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3745 }
```

```
3746 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3747 {
3748   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3749   \dim_zero_new:N \l_@@_width_dim
3750   \dim_set:Nn \l_@@_width_dim { #1 }
3751   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3752   \@@_settings_for_tabular:
3753   \NiceArray { #3 }
```

```

3754 }
3755 {
3756   \endNiceArray
3757   \int_if_zero:nT \g_@@_total_X_weight_int
3758   { \@@_error:n { NiceTabularX~without~X } }
3759 }

3760 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3761 {
3762   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3763   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3764   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3765   \@@_settings_for_tabular:
3766   \NiceArray { #3 }
3767 }
3768 { \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`).

```

3769 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3770 {
3771   \bool_lazy_all:nT
3772   {
3773     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3774     \l_@@_hvlines_bool
3775     { ! \g_@@_delims_bool }
3776     { ! \l_@@_except_borders_bool }
3777   }
3778   {
3779     \bool_set_true:N \l_@@_except_borders_bool
3780     \clist_if_empty:NF \l_@@_corners_clist
3781     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3782     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3783     {
3784       \@@_stroke_block:nnn
3785       {
3786         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3787         draw = \l_@@_rules_color_tl
3788       }
3789       { 1-1 }
3790       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3791     }
3792   }
3793 }

```

```

3794 \cs_new_protected:Npn \@@_after_array:
3795 {
3796   \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.

```

3797   \bool_if:NT \g_@@_last_col_found_bool
3798   { \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`.

```
3799 \bool_if:NT \l_@@_last_col_without_value_bool
3800 { \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.

```
3801 \bool_if:NT \l_@@_last_row_without_value_bool
3802 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3803 \tl_build_gput_right:Nx \g_@@_aux_tl
3804 {
3805   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3806   {
3807     \int_use:N \l_@@_first_row_int ,
3808     \int_use:N \c@iRow ,
3809     \int_use:N \g_@@_row_total_int ,
3810     \int_use:N \l_@@_first_col_int ,
3811     \int_use:N \c@jCol ,
3812     \int_use:N \g_@@_col_total_int
3813   }
3814 }
```

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`).

```
3815 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3816 {
3817   \tl_build_gput_right:Nx \g_@@_aux_tl
3818   {
3819     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3820     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3821   }
3822 }
3823 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3824 {
3825   \tl_build_gput_right:Nx \g_@@_aux_tl
3826   {
3827     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3828     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3829     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3830     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3831   }
3832 }
```

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

```
3833 \@@_create_diag_nodes:
```

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

```
3834 \pgfpicture
3835 \int_step_inline:nn \c@iRow
3836 {
3837   \pgfnodealias
3838   { \@@_env: - ##1 - last }
3839   { \@@_env: - ##1 - \int_use:N \c@jCol }
3840 }
3841 \int_step_inline:nn \c@jCol
3842 {
3843   \pgfnodealias
3844   { \@@_env: - last - ##1 }
3845   { \@@_env: - \int_use:N \c@iRow - ##1 }
3846 }
3847 \str_if_empty:NF \l_@@_name_str
3848 {
```

```

3849 \int_step_inline:nn \c@iRow
3850 {
3851   \pgfnodealias
3852     { \l_@@_name_str - ##1 - last }
3853     { \@@_env: - ##1 - \int_use:N \c@jCol }
3854 }
3855 \int_step_inline:nn \c@jCol
3856 {
3857   \pgfnodealias
3858     { \l_@@_name_str - last - ##1 }
3859     { \@@_env: - \int_use:N \c@iRow - ##1 }
3860 }
3861 }
3862 \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.

```

3863 \bool_if:NT \l_@@_parallelize_diags_bool
3864 {
3865   \int_gzero_new:N \g_@@_ddots_int
3866   \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.

```

3867   \dim_gzero_new:N \g_@@_delta_x_one_dim
3868   \dim_gzero_new:N \g_@@_delta_y_one_dim
3869   \dim_gzero_new:N \g_@@_delta_x_two_dim
3870   \dim_gzero_new:N \g_@@_delta_y_two_dim
3871 }
3872 \int_zero_new:N \l_@@_initial_i_int
3873 \int_zero_new:N \l_@@_initial_j_int
3874 \int_zero_new:N \l_@@_final_i_int
3875 \int_zero_new:N \l_@@_final_j_int
3876 \bool_set_false:N \l_@@_initial_open_bool
3877 \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.

```

3878 \bool_if:NT \l_@@_small_bool
3879 {
3880   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3881   \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.

```

3882   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3883     { 0.6 \l_@@_xdots_shorten_start_dim }
3884   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3885     { 0.6 \l_@@_xdots_shorten_end_dim }
3886 }

```

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

```

3887 \@@_draw_dotted_lines:

```

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

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

```
3888 \@@_compute_corners:
```

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

```
3889 \@@_adjust_pos_of_blocks_seq:
3890 \@@_deal_with_rounded_corners:
3891 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3892 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3893 \IfPackageLoadedTF { tikz }
3894 {
3895   \tikzset
3896   {
3897     every-picture / .style =
3898     {
3899       overlay ,
3900       remember-picture ,
3901       name-prefix = \@@_env: -
3902     }
3903   }
3904 }
3905 { }
3906 \cs_set_eq:NN \ialign \@@_old_ialign:
3907 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3908 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3909 \cs_set_eq:NN \OverBrace \@@_OverBrace
3910 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3911 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3912 \cs_set_eq:NN \line \@@_line
3913 \g_@@_pre_code_after_tl
3914 \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.

```
3915 \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).

```
3916 \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`).

```
3917 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3918 { \@@_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:`.

```
3919 \bool_set_true:N \l_@@_in_code_after_bool
3920 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3921 \scan_stop:
3922 \tl_gclear:N \g_nicematrix_code_after_tl
3923 \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.

```

3924 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3925 \tl_if_empty:NF \g_@@_pre_code_before_tl
3926 {
3927   \tl_build_gput_right:Nx \g_@@_aux_tl
3928   {
3929     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3930     { \exp_not:o \g_@@_pre_code_before_tl }
3931   }
3932   \tl_gclear:N \g_@@_pre_code_before_tl
3933 }
3934 \tl_if_empty:NF \g_nicematrix_code_before_tl
3935 {
3936   \tl_build_gput_right:Nx \g_@@_aux_tl
3937   {
3938     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3939     { \exp_not:o \g_nicematrix_code_before_tl }
3940   }
3941   \tl_gclear:N \g_nicematrix_code_before_tl
3942 }

3943 \str_gclear:N \g_@@_name_env_str
3944 \@@_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`.

```

3945 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3946 }

```

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`.

```

3947 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3948 { \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).

```

3949 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3950 {
3951   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3952   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3953 }

```

The following command must *not* be protected.

```

3954 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3955 {
3956   { #1 }
3957   { #2 }
3958   {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3959     \int_compare:nNnTF { #3 } > { 99 }
3960     { \int_use:N \c@iRow }
3961     { #3 }
3962   }
3963   {
3964     \int_compare:nNnTF { #4 } > { 99 }
3965     { \int_use:N \c@jCol }
3966     { #4 }
3967   }
3968   { #5 }
3969 }

```

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).

```

3970 \hook_gput_code:nnn { begindocument } { . }
3971 {
3972   \cs_new_protected:Npx \@@_draw_dotted_lines:
3973   {
3974     \c_@@_pgfortikzpicture_tl
3975     \@@_draw_dotted_lines_i:
3976     \c_@@_endpgfortikzpicture_tl
3977   }
3978 }

```

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

```

3979 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3980 {
3981   \pgfrememberpicturepositiononpagetrue
3982   \pgf@relevantforpicturesizefalse
3983   \g_@@_HVdotsfor_lines_tl
3984   \g_@@_Vdots_lines_tl
3985   \g_@@_Ddots_lines_tl
3986   \g_@@_Iddots_lines_tl
3987   \g_@@_Cdots_lines_tl
3988   \g_@@_Ldots_lines_tl
3989 }

3990 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3991 {
3992   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3993   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3994 }

```

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

```

3995 \pgfdeclareshape { @@_diag_node }
3996 {
3997   \savedanchor { \five }
3998   {
3999     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4000     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4001   }
4002   \anchor { 5 } { \five }
4003   \anchor { center } { \pgfpointorigin }
4004 }

```

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).

```

4005 \cs_new_protected:Npn \@@_create_diag_nodes:

```

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.

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

17 We draw the dotted lines

$$\begin{pmatrix} a+b+c & a+b & a \\ a & \dots\dots\dots & \\ a & a+b & a+b+c \end{pmatrix}$$

- 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.

```
4046 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4047 {
```

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

```
4048 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4049 \int_set:Nn \l_@@_initial_i_int { #1 }
4050 \int_set:Nn \l_@@_initial_j_int { #2 }
4051 \int_set:Nn \l_@@_final_i_int { #1 }
4052 \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.

```
4053 \bool_set_false:N \l_@@_stop_loop_bool
4054 \bool_do_until:Nn \l_@@_stop_loop_bool
4055 {
4056   \int_add:Nn \l_@@_final_i_int { #3 }
4057   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4058   \bool_set_false:N \l_@@_final_open_bool
4059   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4060   {
4061     \int_compare:nNnTF { #3 } = \c_one_int
4062     { \bool_set_true:N \l_@@_final_open_bool }
4063     {
4064       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4065       { \bool_set_true:N \l_@@_final_open_bool }
4066     }
4067   }
4068   {
4069     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
4070     {
4071       \int_compare:nNnT { #4 } = { -1 }
4072       { \bool_set_true:N \l_@@_final_open_bool }
4073     }
4074     {
4075       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4076       {
4077         \int_compare:nNnT { #4 } = \c_one_int
4078         { \bool_set_true:N \l_@@_final_open_bool }
4079       }
4080     }
4081   }
4082   \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.

```

4083         {
We do a step backwards.
4084         \int_sub:Nn \l_@@_final_i_int { #3 }
4085         \int_sub:Nn \l_@@_final_j_int { #4 }
4086         \bool_set_true:N \l_@@_stop_loop_bool
4087     }

```

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`.

```

4088     {
4089         \cs_if_exist:cTF
4090         {
4091             @@ _ dotted _
4092             \int_use:N \l_@@_final_i_int -
4093             \int_use:N \l_@@_final_j_int
4094         }
4095         {
4096             \int_sub:Nn \l_@@_final_i_int { #3 }
4097             \int_sub:Nn \l_@@_final_j_int { #4 }
4098             \bool_set_true:N \l_@@_final_open_bool
4099             \bool_set_true:N \l_@@_stop_loop_bool
4100         }
4101         {
4102             \cs_if_exist:cTF
4103             {
4104                 pgf @ sh @ ns @ \@@_env:
4105                 - \int_use:N \l_@@_final_i_int
4106                 - \int_use:N \l_@@_final_j_int
4107             }
4108             { \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.

```

4109         {
4110             \cs_set:cpn
4111             {
4112                 @@ _ dotted _
4113                 \int_use:N \l_@@_final_i_int -
4114                 \int_use:N \l_@@_final_j_int
4115             }
4116             { }
4117         }
4118     }
4119 }
4120

```

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

```

4121 \bool_set_false:N \l_@@_stop_loop_bool
4122 \bool_do_until:Nn \l_@@_stop_loop_bool
4123 {
4124     \int_sub:Nn \l_@@_initial_i_int { #3 }
4125     \int_sub:Nn \l_@@_initial_j_int { #4 }
4126     \bool_set_false:N \l_@@_initial_open_bool
4127     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4128     {
4129         \int_compare:nNnTF { #3 } = \c_one_int
4130         { \bool_set_true:N \l_@@_initial_open_bool }

```

```

4131     {
4132         \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4133         { \bool_set_true:N \l_@@_initial_open_bool }
4134     }
4135 }
4136 {
4137     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4138     {
4139         \int_compare:nNnT { #4 } = \c_one_int
4140         { \bool_set_true:N \l_@@_initial_open_bool }
4141     }
4142     {
4143         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4144         {
4145             \int_compare:nNnT { #4 } = { -1 }
4146             { \bool_set_true:N \l_@@_initial_open_bool }
4147         }
4148     }
4149 }
4150 \bool_if:NTF \l_@@_initial_open_bool
4151 {
4152     \int_add:Nn \l_@@_initial_i_int { #3 }
4153     \int_add:Nn \l_@@_initial_j_int { #4 }
4154     \bool_set_true:N \l_@@_stop_loop_bool
4155 }
4156 {
4157     \cs_if_exist:cTF
4158     {
4159         @@ _ dotted _
4160         \int_use:N \l_@@_initial_i_int -
4161         \int_use:N \l_@@_initial_j_int
4162     }
4163     {
4164         \int_add:Nn \l_@@_initial_i_int { #3 }
4165         \int_add:Nn \l_@@_initial_j_int { #4 }
4166         \bool_set_true:N \l_@@_initial_open_bool
4167         \bool_set_true:N \l_@@_stop_loop_bool
4168     }
4169     {
4170         \cs_if_exist:cTF
4171         {
4172             pgf @ sh @ ns @ \@@_env:
4173             - \int_use:N \l_@@_initial_i_int
4174             - \int_use:N \l_@@_initial_j_int
4175         }
4176         { \bool_set_true:N \l_@@_stop_loop_bool }
4177         {
4178             \cs_set:cpn
4179             {
4180                 @@ _ dotted _
4181                 \int_use:N \l_@@_initial_i_int -
4182                 \int_use:N \l_@@_initial_j_int
4183             }
4184             { }
4185         }
4186     }
4187 }
4188 }

```

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.

```

4189 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4190 {
4191     { \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`.

```

4192     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4193     { \int_use:N \l_@@_final_i_int }
4194     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4195     { } % for the name of the block
4196   }
4197 }

```

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 analysis 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.

```

4198 \cs_new_protected:Npn \@@_open_shorten:
4199 {
4200   \bool_if:NT \l_@@_initial_open_bool
4201   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4202   \bool_if:NT \l_@@_final_open_bool
4203   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4204 }

```

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).

```

4205 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4206 {
4207   \int_set:Nn \l_@@_row_min_int 1
4208   \int_set:Nn \l_@@_col_min_int 1
4209   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4210   \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`.

```

4211 \seq_map_inline:Nn \g_@@_submatrix_seq
4212 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4213 }

```

#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.

```

4214 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4215 {
4216   \int_compare:nNnF { #3 } > { #1 }
4217   {
4218     \int_compare:nNnF { #1 } > { #5 }
4219     {
4220       \int_compare:nNnF { #4 } > { #2 }
4221       {
4222         \int_compare:nNnF { #2 } > { #6 }
4223         {
4224           \int_set:Nn \l_@@_row_min_int
4225           { \int_max:nn \l_@@_row_min_int { #3 } }
4226           \int_set:Nn \l_@@_col_min_int
4227           { \int_max:nn \l_@@_col_min_int { #4 } }
4228           \int_set:Nn \l_@@_row_max_int
4229           { \int_min:nn \l_@@_row_max_int { #5 } }
4230           \int_set:Nn \l_@@_col_max_int
4231           { \int_min:nn \l_@@_col_max_int { #6 } }
4232         }

```

```

4233     }
4234   }
4235 }
4236 }

4237 \cs_new_protected:Npn \@@_set_initial_coords:
4238 {
4239   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4240   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4241 }
4242 \cs_new_protected:Npn \@@_set_final_coords:
4243 {
4244   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4245   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4246 }
4247 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4248 {
4249   \pgfpointanchor
4250   {
4251     \@@_env:
4252     - \int_use:N \l_@@_initial_i_int
4253     - \int_use:N \l_@@_initial_j_int
4254   }
4255   { #1 }
4256   \@@_set_initial_coords:
4257 }
4258 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4259 {
4260   \pgfpointanchor
4261   {
4262     \@@_env:
4263     - \int_use:N \l_@@_final_i_int
4264     - \int_use:N \l_@@_final_j_int
4265   }
4266   { #1 }
4267   \@@_set_final_coords:
4268 }

4269 \cs_new_protected:Npn \@@_open_x_initial_dim:
4270 {
4271   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4272   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4273   {
4274     \cs_if_exist:cT
4275     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4276     {
4277       \pgfpointanchor
4278       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4279       { west }
4280       \dim_set:Nn \l_@@_x_initial_dim
4281       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4282     }
4283   }

```

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

```

4284   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4285   {
4286     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4287     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4288     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4289   }
4290 }

4291 \cs_new_protected:Npn \@@_open_x_final_dim:
4292 {

```



```

4293 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4294 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4295 {
4296   \cs_if_exist:cT
4297   { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4298   {
4299     \pgfpointanchor
4300     { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4301     { east }
4302     \dim_set:Nn \l_@@_x_final_dim
4303     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4304   }
4305 }

```

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

```

4306 \dim_compare:nNtT \l_@@_x_final_dim = { - \c_max_dim }
4307 {
4308   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4309   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4310   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4311 }
4312 }

```

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.

```

4313 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4314 {
4315   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4316   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4317   {
4318     \@@_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.

```

4319   \group_begin:
4320   \@@_open_shorten:
4321   \int_if_zero:nTF { #1 }
4322   { \color { nicematrix-first-row } }
4323   {

```

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.

```

4324     \int_compare:nNtT { #1 } = \l_@@_last_row_int
4325     { \color { nicematrix-last-row } }
4326   }
4327   \keys_set:nn { NiceMatrix / xdots } { #3 }
4328   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4329   \@@_actually_draw_Ldots:
4330   \group_end:
4331 }
4332 }

```

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`.

```

4333 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4334 {
4335   \bool_if:NTF \l_@@_initial_open_bool
4336   {
4337     \@@_open_x_initial_dim:
4338     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4339     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4340   }
4341   { \@@_set_initial_coords_from_anchor:n { base-east } }
4342 \bool_if:NTF \l_@@_final_open_bool
4343 {
4344   \@@_open_x_final_dim:
4345   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4346   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4347 }
4348 { \@@_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.

```

4349 \bool_lazy_all:nTF
4350 {
4351   \l_@@_initial_open_bool
4352   \l_@@_final_open_bool
4353   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4354 }
4355 {
4356   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4357   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4358 }

```

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 (?).

```

4359 {
4360   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4361   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4362 }
4363 \@@_draw_line:
4364 }

```

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.

```

4365 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4366 {
4367   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4368   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4369   {
4370     \@@_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.

```

4371 \group_begin:
4372   \@@_open_shorten:
4373   \int_if_zero:nTF { #1 }
4374   { \color { nicematrix-first-row } }
4375   {

```

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.

```

4376   \int_compare:nNnT { #1 } = \l_@@_last_row_int
4377   { \color { nicematrix-last-row } }
4378 }

```

```

4379         \keys_set:nn { NiceMatrix / xdots } { #3 }
4380         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4381         \@@_actually_draw_Cdots:
4382     \group_end:
4383 }
4384 }

```

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.`

```

4385 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4386 {
4387     \bool_if:NTF \l_@@_initial_open_bool
4388     { \@@_open_x_initial_dim: }
4389     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4390     \bool_if:NTF \l_@@_final_open_bool
4391     { \@@_open_x_final_dim: }
4392     { \@@_set_final_coords_from_anchor:n { mid-west } }
4393     \bool_lazy_and:nnTF
4394     \l_@@_initial_open_bool
4395     \l_@@_final_open_bool
4396     {
4397         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4398         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4399         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4400         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4401         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4402     }
4403     {
4404         \bool_if:NT \l_@@_initial_open_bool
4405         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4406         \bool_if:NT \l_@@_final_open_bool
4407         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4408     }
4409     \@@_draw_line:
4410 }
4411 \cs_new_protected:Npn \@@_open_y_initial_dim:
4412 {
4413     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4414     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4415     {
4416         \cs_if_exist:cT
4417         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4418         {
4419             \pgfpointanchor
4420             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4421             { north }
4422             \dim_set:Nn \l_@@_y_initial_dim
4423             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4424         }
4425     }
4426     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4427     {
4428         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }

```

```

4429 \dim_set:Nn \l_@@_y_initial_dim
4430 {
4431   \fp_to_dim:n
4432   {
4433     \pgf@y
4434     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4435   }
4436 }
4437 }
4438 }
4439 \cs_new_protected:Npn \@@_open_y_final_dim:
4440 {
4441   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4442   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4443   {
4444     \cs_if_exist:cT
4445     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4446     {
4447       \pgfpointanchor
4448       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4449       { south }
4450       \dim_set:Nn \l_@@_y_final_dim
4451       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4452     }
4453   }
4454   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4455   {
4456     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4457     \dim_set:Nn \l_@@_y_final_dim
4458     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4459   }
4460 }

```

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.

```

4461 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4462 {
4463   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4464   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4465   {
4466     \@@_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.

```

4467 \group_begin:
4468   \@@_open_shorten:
4469   \int_if_zero:nTF { #2 }
4470   { \color { nicematrix-first-col } }
4471   {
4472     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4473     { \color { nicematrix-last-col } }
4474   }
4475   \keys_set:nn { NiceMatrix / xdots } { #3 }
4476   \tl_if_empty:oF \l_@@_xdots_color_tl
4477   { \color { \l_@@_xdots_color_tl } }
4478   \@@_actually_draw_Vdots:
4479 \group_end:
4480 }
4481 }

```

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.

```
4482 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4483 {
```

First, the case of a dotted line open on both sides.

```
4484 \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.

```
4485 {
4486   \@@_open_y_initial_dim:
4487   \@@_open_y_final_dim:
4488   \int_if_zero:nTF \l_@@_initial_j_int
```

We have a dotted line open on both sides in the “first column”.

```
4489 {
4490   \@@_qpoint:n { col - 1 }
4491   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4492   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4493   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4494   \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4495 }
4496 {
4497   \bool_lazy_and:nnTF
4498   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4499   { \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”.

```
4500 {
4501   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4502   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4503   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4504   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4505   \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4506 }
```

We have a dotted line open on both sides which is *not* in an exterior column.

```
4507 {
4508   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4509   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4510   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4511   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4512 }
4513 }
4514 }
```

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.

```
4515 {
4516   \bool_set_false:N \l_tmpa_bool
4517   \bool_if:NF \l_@@_initial_open_bool
4518   {
4519     \bool_if:NF \l_@@_final_open_bool
4520     {
4521       \@@_set_initial_coords_from_anchor:n { south-west }
4522       \@@_set_final_coords_from_anchor:n { north-west }
4523       \bool_set:Nn \l_tmpa_bool
4524       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
```

```

4525     }
4526 }

```

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

```

4527 \bool_if:NTF \l_@@_initial_open_bool
4528 {
4529   \@@_open_y_initial_dim:
4530   \@@_set_final_coords_from_anchor:n { north }
4531   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4532 }
4533 {
4534   \@@_set_initial_coords_from_anchor:n { south }
4535   \bool_if:NTF \l_@@_final_open_bool
4536   \@@_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.

```

4537 {
4538   \@@_set_final_coords_from_anchor:n { north }
4539   \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4540   {
4541     \dim_set:Nn \l_@@_x_initial_dim
4542     {
4543       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4544       \l_@@_x_initial_dim \l_@@_x_final_dim
4545     }
4546   }
4547 }
4548 }
4549 }
4550 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4551 \@@_draw_line:
4552 }

```

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.

```

4553 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4554 {
4555   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4556   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4557   {
4558     \@@_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.

```

4559 \group_begin:
4560 \@@_open_shorten:
4561 \keys_set:nn { NiceMatrix / xdots } { #3 }
4562 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4563 \@@_actually_draw_Ddots:
4564 \group_end:
4565 }
4566 }

```

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.

```

4567 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4568 {
4569   \bool_if:NTF \l_@@_initial_open_bool
4570   {
4571     \@@_open_y_initial_dim:
4572     \@@_open_x_initial_dim:
4573   }
4574   { \@@_set_initial_coords_from_anchor:n { south~east } }
4575   \bool_if:NTF \l_@@_final_open_bool
4576   {
4577     \@@_open_x_final_dim:
4578     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4579   }
4580   { \@@_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.

```

4581   \bool_if:NT \l_@@_parallelize_diags_bool
4582   {
4583     \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).

```

4584     \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.

```

4585     {
4586       \dim_gset:Nn \g_@@_delta_x_one_dim
4587       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4588       \dim_gset:Nn \g_@@_delta_y_one_dim
4589       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4590     }

```

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.

```

4591     {
4592       \dim_set:Nn \l_@@_y_final_dim
4593       {
4594         \l_@@_y_initial_dim +
4595         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4596         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4597       }
4598     }
4599   }
4600   \@@_draw_line:
4601 }

```

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.

```

4602 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4603 {
4604   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4605   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4606   {
4607     \@@_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.

```

4608     \group_begin:
4609     \@@_open_shorten:
4610     \keys_set:nn { NiceMatrix / xdots } { #3 }
4611     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4612     \@@_actually_draw_Iddots:
4613     \group_end:
4614   }
4615 }

```

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.`

```

4616 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4617 {
4618   \bool_if:NTF \l_@@_initial_open_bool
4619   {
4620     \@@_open_y_initial_dim:
4621     \@@_open_x_initial_dim:
4622   }
4623   { \@@_set_initial_coords_from_anchor:n { south-west } }
4624   \bool_if:NTF \l_@@_final_open_bool
4625   {
4626     \@@_open_y_final_dim:
4627     \@@_open_x_final_dim:
4628   }
4629   { \@@_set_final_coords_from_anchor:n { north-east } }
4630   \bool_if:NT \l_@@_parallelize_diags_bool
4631   {
4632     \int_gincr:N \g_@@_iddots_int
4633     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4634     {
4635       \dim_gset:Nn \g_@@_delta_x_two_dim
4636       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4637       \dim_gset:Nn \g_@@_delta_y_two_dim
4638       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4639     }
4640     {
4641       \dim_set:Nn \l_@@_y_final_dim
4642       {
4643         \l_@@_y_initial_dim +
4644         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4645         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4646       }
4647     }
4648   }
4649   \@@_draw_line:
4650 }

```


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`

```

4651 \cs_new_protected:Npn \@@_draw_line:
4652 {
4653   \pgfrememberpicturepositiononpagetrue
4654   \pgf@relevantforpicturesizefalse
4655   \bool_lazy_or:nnTF
4656     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4657     \l_@@_dotted_bool
4658     \@@_draw_standard_dotted_line:
4659     \@@_draw_unstandard_dotted_line:
4660 }

```

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.

```

4661 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4662 {
4663   \begin { scope }
4664   \@@_draw_unstandard_dotted_line:o
4665     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4666 }

```

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.

```

4667 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4668 {
4669   \@@_draw_unstandard_dotted_line:nooo
4670     { #1 }
4671     \l_@@_xdots_up_tl
4672     \l_@@_xdots_down_tl
4673     \l_@@_xdots_middle_tl
4674 }
4675 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4676 \hook_gput_code:nnn { begindocument } { . }
4677 {
4678   \IfPackageLoadedTF { tikz }
4679   {
4680     \tikzset
4681     {
4682       @@_node_above / .style = { sloped , above } ,
4683       @@_node_below / .style = { sloped , below } ,
4684       @@_node_middle / .style =
4685       {

```

```

4686         sloped ,
4687         inner~sep = \c_@@_innersep_middle_dim
4688     }
4689 }
4690 }
4691 { }
4692 }

```

```

4693 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4694 {

```

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.

```

4695     \dim_zero_new:N \l_@@_l_dim
4696     \dim_set:Nn \l_@@_l_dim
4697     {
4698         \fp_to_dim:n
4699         {
4700             sqrt
4701             (
4702                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4703                 +
4704                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4705             )
4706         }
4707     }

```

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.

```

4708     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4709     {
4710         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4711         \@@_draw_unstandard_dotted_line_i:
4712     }

```

If the key `xdots/horizontal-labels` has been used.

```

4713     \bool_if:NT \l_@@_xdots_h_labels_bool
4714     {
4715         \tikzset
4716         {
4717             @@_node_above / .style = { auto = left } ,
4718             @@_node_below / .style = { auto = right } ,
4719             @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4720         }
4721     }
4722     \tl_if_empty:nF { #4 }
4723     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4724     \draw
4725     [ #1 ]
4726     ( \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).

```

4727         -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4728         node [ @@_node_below ] { $ \scriptstyle #3 $ }
4729         node [ @@_node_above ] { $ \scriptstyle #2 $ }
4730         ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4731     \end { scope }
4732 }

```

```

4733 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4734 {
4735   \dim_set:Nn \l_tmpa_dim
4736   {
4737     \l_@@_x_initial_dim
4738     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4739     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4740   }
4741   \dim_set:Nn \l_tmpb_dim
4742   {
4743     \l_@@_y_initial_dim
4744     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4745     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4746   }
4747   \dim_set:Nn \l_@@_tmpc_dim
4748   {
4749     \l_@@_x_final_dim
4750     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4751     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4752   }
4753   \dim_set:Nn \l_@@_tmpd_dim
4754   {
4755     \l_@@_y_final_dim
4756     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4757     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4758   }
4759   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4760   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4761   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4762   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4763 }
4764 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4765 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4766 {
4767   \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.

```

4768   \dim_zero_new:N \l_@@_l_dim
4769   \dim_set:Nn \l_@@_l_dim
4770   {
4771     \fp_to_dim:n
4772     {
4773       sqrt
4774       (
4775         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4776         +
4777         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4778       )
4779     }
4780   }

```

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.

```

4781   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4782   {
4783     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4784     \@@_draw_standard_dotted_line_i:

```

```

4785     }
4786 \group_end:
4787 \bool_lazy_all:nF
4788 {
4789   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4790   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4791   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4792 }
4793 \l_@@_labels_standard_dotted_line:
4794 }
4795 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4796 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4797 {

```

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

```

4798   \int_set:Nn \l_tmpa_int
4799   {
4800     \dim_ratio:nn
4801     {
4802       \l_@@_l_dim
4803       - \l_@@_xdots_shorten_start_dim
4804       - \l_@@_xdots_shorten_end_dim
4805     }
4806     \l_@@_xdots_inter_dim
4807   }

```

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

```

4808   \dim_set:Nn \l_tmpa_dim
4809   {
4810     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4811     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4812   }
4813   \dim_set:Nn \l_tmpb_dim
4814   {
4815     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4816     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4817   }

```

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.

```

4818   \dim_gadd:Nn \l_@@_x_initial_dim
4819   {
4820     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4821     \dim_ratio:nn
4822     {
4823       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4824       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4825     }
4826     { 2 \l_@@_l_dim }
4827   }
4828   \dim_gadd:Nn \l_@@_y_initial_dim
4829   {
4830     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4831     \dim_ratio:nn
4832     {
4833       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4834       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4835     }
4836     { 2 \l_@@_l_dim }
4837   }
4838   \pgf@relevantforpicturesizefalse

```

```

4839 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4840 {
4841   \pgfpathcircle
4842     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4843     { \l_@@_xdots_radius_dim }
4844   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4845   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4846 }
4847 \pgfusepathqfill
4848 }

4849 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4850 {
4851   \pgfscope
4852   \pgftransformshift
4853   {
4854     \pgfpointlineattime { 0.5 }
4855     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4856     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4857   }
4858   \fp_set:Nn \l_tmpa_fp
4859   {
4860     atand
4861     (
4862       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4863       \l_@@_x_final_dim - \l_@@_x_initial_dim
4864     )
4865   }
4866   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4867   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4868   \tl_if_empty:NF \l_@@_xdots_middle_tl
4869   {
4870     \begin { pgfscope }
4871     \pgfset { inner-sep = \c_@@_innersep_middle_dim }
4872     \pgfnode
4873       { rectangle }
4874       { center }
4875       {
4876         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4877         {
4878           \c_math_toggle_token
4879           \scriptstyle \l_@@_xdots_middle_tl
4880           \c_math_toggle_token
4881         }
4882       }
4883     { }
4884     {
4885       \pgfsetfillcolor { white }
4886       \pgfusepath { fill }
4887     }
4888     \end { pgfscope }
4889   }
4890   \tl_if_empty:NF \l_@@_xdots_up_tl
4891   {
4892     \pgfnode
4893       { rectangle }
4894       { south }
4895       {
4896         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4897         {
4898           \c_math_toggle_token
4899           \scriptstyle \l_@@_xdots_up_tl
4900           \c_math_toggle_token

```

```

4901         }
4902     }
4903     { }
4904     { \pgfusepath { } }
4905 }
4906 \tl_if_empty:NF \l_@@_xdots_down_tl
4907 {
4908     \pgfnode
4909     { rectangle }
4910     { north }
4911     {
4912         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4913         {
4914             \c_math_toggle_token
4915             \scriptstyle \l_@@_xdots_down_tl
4916             \c_math_toggle_token
4917         }
4918     }
4919     { }
4920     { \pgfusepath { } }
4921 }
4922 \endpgfscope
4923 }

```

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.

```

4924 \hook_gput_code:nnn { begindocument } { . }
4925 {
4926     \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4927     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4928     \cs_new_protected:Npn \@@_Ldots
4929     { \@@_collect_options:n { \@@_Ldots_i } }
4930     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4931     {
4932         \int_if_zero:nTF \c@jCol
4933         { \@@_error:nn { in~first~col } \Ldots }
4934         {
4935             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4936             { \@@_error:nn { in~last~col } \Ldots }
4937             {
4938                 \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4939                 { #1 , down = #2 , up = #3 , middle = #4 }
4940             }
4941         }
4942         \bool_if:NF \l_@@_nullify_dots_bool
4943         { \phantom { \ensuremath { \@@_old_ldots } } }
4944         \bool_gset_true:N \g_@@_empty_cell_bool
4945     }

```

```

4946 \cs_new_protected:Npn \@@_Cdots
4947 { \@@_collect_options:n { \@@_Cdots_i } }
4948 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4949 {
4950   \int_if_zero:nTF \c@jCol
4951   { \@@_error:nn { in~first~col } \Cdots }
4952   {
4953     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4954     { \@@_error:nn { in~last~col } \Cdots }
4955     {
4956       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4957       { #1 , down = #2 , up = #3 , middle = #4 }
4958     }
4959   }
4960   \bool_if:NF \l_@@_nullify_dots_bool
4961   { \phantom { \ensuremath { \@@_old_cdots } } }
4962   \bool_gset_true:N \g_@@_empty_cell_bool
4963 }

4964 \cs_new_protected:Npn \@@_Vdots
4965 { \@@_collect_options:n { \@@_Vdots_i } }
4966 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4967 {
4968   \int_if_zero:nTF \c@iRow
4969   { \@@_error:nn { in~first~row } \Vdots }
4970   {
4971     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4972     { \@@_error:nn { in~last~row } \Vdots }
4973     {
4974       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4975       { #1 , down = #2 , up = #3 , middle = #4 }
4976     }
4977   }
4978   \bool_if:NF \l_@@_nullify_dots_bool
4979   { \phantom { \ensuremath { \@@_old_vdots } } }
4980   \bool_gset_true:N \g_@@_empty_cell_bool
4981 }

4982 \cs_new_protected:Npn \@@_Ddots
4983 { \@@_collect_options:n { \@@_Ddots_i } }
4984 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4985 {
4986   \int_case:nnF \c@iRow
4987   {
4988     0 { \@@_error:nn { in~first~row } \Ddots }
4989     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4990   }
4991   {
4992     \int_case:nnF \c@jCol
4993     {
4994       0 { \@@_error:nn { in~first~col } \Ddots }
4995       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4996     }
4997     {
4998       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4999       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5000       { #1 , down = #2 , up = #3 , middle = #4 }
5001     }
5002   }
5003   \bool_if:NF \l_@@_nullify_dots_bool
5004   { \phantom { \ensuremath { \@@_old_ddots } } }
5005 }

```

```

5006     \bool_gset_true:N \g_@@_empty_cell_bool
5007 }

5008 \cs_new_protected:Npn \@@_Iddots
5009 { \@@_collect_options:n { \@@_Iddots_i } }
5010 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5011 {
5012     \int_case:nnF \c@iRow
5013     {
5014         0 { \@@_error:nn { in~first~row } \Iddots }
5015         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5016     }
5017     {
5018         \int_case:nnF \c@jCol
5019         {
5020             0 { \@@_error:nn { in~first~col } \Iddots }
5021             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5022         }
5023         {
5024             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5025             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5026             { #1 , down = #2 , up = #3 , middle = #4 }
5027         }
5028     }
5029     \bool_if:NF \l_@@_nullify_dots_bool
5030     { \phantom { \ensuremath { \@@_old_iddots } } }
5031     \bool_gset_true:N \g_@@_empty_cell_bool
5032 }
5033 }

```

End of the \AddToHook.

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

```

5034 \keys_define:nn { NiceMatrix / Ddots }
5035 {
5036     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5037     draw-first .default:n = true ,
5038     draw-first .value_forbidden:n = true
5039 }

```

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

```

5040 \cs_new_protected:Npn \@@_Hspace:
5041 {
5042     \bool_gset_true:N \g_@@_empty_cell_bool
5043     \hspace
5044 }

```

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`.

```

5045 \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.

```

5046 \cs_new:Npn \@@_Hdotsfor:
5047 {
5048     \bool_lazy_and:nnTF
5049     { \int_if_zero_p:n \c@jCol }
5050     { \int_if_zero_p:n \l_@@_first_col_int }
5051     {

```



```

5052     \bool_if:NTF \g_@@_after_col_zero_bool
5053     {
5054         \multicolumn { 1 } { c } { }
5055         \@@_Hdotsfor_i
5056     }
5057     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5058 }
5059 {
5060     \multicolumn { 1 } { c } { }
5061     \@@_Hdotsfor_i
5062 }
5063 }

```

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:`).

```

5064 \hook_gput_code:nnn { begindocument } { . }
5065 {
5066     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5067     \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.

```

5068     \cs_new_protected:Npn \@@_Hdotsfor_i
5069     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5070     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5071     {
5072         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5073         {
5074             \@@_Hdotsfor:nnnn
5075             { \int_use:N \c@iRow }
5076             { \int_use:N \c@jCol }
5077             { #2 }
5078             {
5079                 #1 , #3 ,
5080                 down = \exp_not:n { #4 } ,
5081                 up = \exp_not:n { #5 } ,
5082                 middle = \exp_not:n { #6 }
5083             }
5084         }
5085         \prg_replicate:nn { #2 - 1 }
5086         {
5087             &
5088             \multicolumn { 1 } { c } { }
5089             \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5090         }
5091     }
5092 }

```

```

5093 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5094 {
5095     \bool_set_false:N \l_@@_initial_open_bool
5096     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5097     \int_set:Nn \l_@@_initial_i_int { #1 }
5098     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5099     \int_compare:nNnTF { #2 } = \c_one_int
5100     {
5101         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5102         \bool_set_true:N \l_@@_initial_open_bool
5103     }
5104     {

```

```

5105     \cs_if_exist:cTF
5106     {
5107         pgf @ sh @ ns @ \@@_env:
5108         - \int_use:N \l_@@_initial_i_int
5109         - \int_eval:n { #2 - 1 }
5110     }
5111     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5112     {
5113         \int_set:Nn \l_@@_initial_j_int { #2 }
5114         \bool_set_true:N \l_@@_initial_open_bool
5115     }
5116 }
5117 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5118 {
5119     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5120     \bool_set_true:N \l_@@_final_open_bool
5121 }
5122 {
5123     \cs_if_exist:cTF
5124     {
5125         pgf @ sh @ ns @ \@@_env:
5126         - \int_use:N \l_@@_final_i_int
5127         - \int_eval:n { #2 + #3 }
5128     }
5129     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5130     {
5131         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5132         \bool_set_true:N \l_@@_final_open_bool
5133     }
5134 }
5135 \group_begin:
5136 \@@_open_shorten:
5137 \int_if_zero:nTF { #1 }
5138 { \color { nicematrix-first-row } }
5139 {
5140     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5141     { \color { nicematrix-last-row } }
5142 }
5143
5144 \keys_set:nn { NiceMatrix / xdots } { #4 }
5145 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5146 \@@_actually_draw_ldots:
5147 \group_end:

5148     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5149     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5150 }

5151 \hook_gput_code:nnn { begindocument } { . }
5152 {
5153     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5154     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5155     \cs_new_protected:Npn \@@_Vdotsfor:
5156     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5157     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5158     {
5159         \bool_gset_true:N \g_@@_empty_cell_bool
5160         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5161         {

```

```

5162         \@@_Vdotsfor:nnnn
5163         { \int_use:N \c@iRow }
5164         { \int_use:N \c@jCol }
5165         { #2 }
5166         {
5167             #1 , #3 ,
5168             down = \exp_not:n { #4 } ,
5169             up = \exp_not:n { #5 } ,
5170             middle = \exp_not:n { #6 }
5171         }
5172     }
5173 }
5174 }

```

```

5175 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5176 {
5177     \bool_set_false:N \l_@@_initial_open_bool
5178     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5179     \int_set:Nn \l_@@_initial_j_int { #2 }
5180     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5181     \int_compare:nNnTF { #1 } = \c_one_int
5182     {
5183         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5184         \bool_set_true:N \l_@@_initial_open_bool
5185     }
5186     {
5187         \cs_if_exist:cTF
5188         {
5189             pgf @ sh @ ns @ \@@_env:
5190             - \int_eval:n { #1 - 1 }
5191             - \int_use:N \l_@@_initial_j_int
5192         }
5193         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5194         {
5195             \int_set:Nn \l_@@_initial_i_int { #1 }
5196             \bool_set_true:N \l_@@_initial_open_bool
5197         }
5198     }
5199     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5200     {
5201         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5202         \bool_set_true:N \l_@@_final_open_bool
5203     }
5204     {
5205         \cs_if_exist:cTF
5206         {
5207             pgf @ sh @ ns @ \@@_env:
5208             - \int_eval:n { #1 + #3 }
5209             - \int_use:N \l_@@_final_j_int
5210         }
5211         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5212         {
5213             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5214             \bool_set_true:N \l_@@_final_open_bool
5215         }
5216     }
5217     \group_begin:
5218     \@@_open_shorten:
5219     \int_if_zero:nTF { #2 }

```

```

5220     { \color { nicematrix-first-col } }
5221     {
5222         \int_compare:nNt { #2 } = \g_@@_col_total_int
5223         { \color { nicematrix-last-col } }
5224     }
5225     \keys_set:nn { NiceMatrix / xdots } { #4 }
5226     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5227     \@@_actually_draw_Vdots:
5228     \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).

```

5229     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5230     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5231 }

```

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

```

5232 \NewDocumentCommand \@@_rotate: { 0 { } }
5233 {
5234     \peek_remove_spaces:n
5235     {
5236         \bool_gset_true:N \g_@@_rotate_bool
5237         \keys_set:nn { NiceMatrix / rotate } { #1 }
5238     }
5239 }

5240 \keys_define:nn { NiceMatrix / rotate }
5241 {
5242     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5243     c .value_forbidden:n = true ,
5244     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5245 }

```

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

```

5246 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5247 {
5248     \tl_if_empty:nTF { #2 }
5249     { #1 }

```

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

```

5250 { \@@_double_int_eval_i:n #1-#2 \q_stop }
5251 }
5252 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5253 { \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).

```

5254 \hook_gput_code:nnn { begindocument } { . }
5255 {
5256   \cs_set_nopar:Npn \l_@@_argspec_tl
5257   { 0 { } m ! 0 { } E { _ ^ : } { { } { } { } } }
5258   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5259   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5260   {
5261     \group_begin:
5262     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5263     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5264     \use:e
5265     {
5266       \@@_line_i:nn
5267       { \@@_double_int_eval:n #2 - \q_stop }
5268       { \@@_double_int_eval:n #3 - \q_stop }
5269     }
5270     \group_end:
5271   }
5272 }
5273 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5274 {
5275   \bool_set_false:N \l_@@_initial_open_bool
5276   \bool_set_false:N \l_@@_final_open_bool
5277   \bool_lazy_or:nnTF
5278   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5279   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5280   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5281 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5282 }
5283 \hook_gput_code:nnn { begindocument } { . }
5284 {
5285   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5286   {

```

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:`.

```

5287 \c_@@_pgfortikzpicture_tl
5288 \@@_draw_line_iii:nn { #1 } { #2 }
5289 \c_@@_endpgfortikzpicture_tl
5290 }
5291 }

```

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

```

5292 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5293 {
5294   \pgfrememberpicturepositiononpagetrue
5295   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5296   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5297   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5298   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5299   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x

```

```

5300     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5301     \@@_draw_line:
5302 }

```

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`

```

5303 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5304 { \int_compare:nNtT \c@iRow < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```

5305 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5306 {
5307     \tl_gput_right:Nx \g_@@_row_style_tl
5308     {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5309     \exp_not:N
5310     \@@_if_row_less_than:nn
5311     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5312     { \exp_not:n { #1 } }
5313 }
5314 }
5315 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5316 \keys_define:nn { NiceMatrix / RowStyle }
5317 {
5318     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5319     cell-space-top-limit .initial:n = \c_zero_dim ,
5320     cell-space-top-limit .value_required:n = true ,
5321     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5322     cell-space-bottom-limit .initial:n = \c_zero_dim ,
5323     cell-space-bottom-limit .value_required:n = true ,
5324     cell-space-limits .meta:n =
5325     {
5326         cell-space-top-limit = #1 ,
5327         cell-space-bottom-limit = #1 ,
5328     } ,
5329     color .tl_set:N = \l_@@_color_tl ,
5330     color .value_required:n = true ,
5331     bold .bool_set:N = \l_tmpa_bool ,
5332     bold .default:n = true ,
5333     bold .initial:n = false ,
5334     nb-rows .code:n =
5335     \str_if_eq:nnTF { #1 } { * }

```

```

5336     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5337     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5338     nb-rows .value_required:n = true ,
5339     rowcolor .tl_set:N = \l_tmpa_tl ,
5340     rowcolor .value_required:n = true ,
5341     rowcolor .initial:n = ,
5342     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5343 }

```

```

5344 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5345 {
5346   \group_begin:
5347   \tl_clear:N \l_tmpa_tl % value of \rowcolor
5348   \tl_clear:N \l_@@_color_tl
5349   \int_set:Nn \l_@@_key_nb_rows_int 1
5350   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

5351   \tl_if_empty:NF \l_tmpa_tl
5352   {

```

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

```

5353     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5354     {

```

The command \@@_exp_color_arg:No is *fully expandable*.

```

5355         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5356         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5357         { \int_use:N \c@iRow - * }
5358     }

```

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

```

5359     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5360     {
5361       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5362       {
5363         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5364         {
5365           \int_eval:n { \c@iRow + 1 }
5366           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5367         }
5368       }
5369     }
5370   }
5371   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5372   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5373   {
5374     \@@_put_in_row_style:n
5375     {
5376       \exp_not:n
5377       {
5378         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
5379         {
5380           \dim_set:Nn \l_@@_cell_space_top_limit_dim
5381           { \dim_use:N \l_tmpa_dim }
5382         }
5383       }
5384     }
5385   }

```

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

```

5386   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5387   {
5388     \@@_put_in_row_style:n
5389     {
5390       \exp_not:n
5391       {
5392         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
5393         {
5394           \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5395           { \dim_use:N \l_tmpb_dim }
5396         }
5397       }
5398     }
5399   }

```

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

```

5400   \tl_if_empty:NF \l_@@_color_tl
5401   {
5402     \@@_put_in_row_style:e
5403     {
5404       \mode_leave_vertical:
5405       \@@_color:n { \l_@@_color_tl }
5406     }
5407   }

```

`\l_tmpa_bool` is the value of the key `bold`.

```

5408   \bool_if:NT \l_tmpa_bool
5409   {
5410     \@@_put_in_row_style:n
5411     {
5412       \exp_not:n
5413       {
5414         \if_mode_math:
5415         \c_math_toggle_token
5416         \bfseries \boldmath
5417         \c_math_toggle_token
5418       \else:
5419         \bfseries \boldmath
5420       \fi:
5421     }
5422   }
5423 }
5424 \group_end:
5425 \g_@@_row_style_tl
5426 \ignorespaces
5427 }

```

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).

```
5428 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5429 {
```

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.

```
5430 \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`.

```
5431 \str_if_in:nnF { #1 } { !! }
5432 {
5433   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5434   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##2 } } }
5435 }
5436 \int_if_zero:nTF \l_tmpa_int
```

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

```
5437 {
5438   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5439   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5440 }
```

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

```
5441 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5442 }
5443 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5444 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5445 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5446 {
5447   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5448   {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5449 \group_begin:
5450 \pgfsetcornersarced
5451 {
5452   \pgfpoint
5453   { \l_@@_tab_rounded_corners_dim }
5454   { \l_@@_tab_rounded_corners_dim }
5455 }
```

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.

```
5456 \bool_if:NTF \l_@@_hvlines_bool
5457 {
5458   \pgfpathrectanglecorners
5459   {
```

```

5460         \pgfpointadd
5461         { \@@_qpoint:n { row-1 } }
5462         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5463     }
5464     {
5465         \pgfpointadd
5466         {
5467             \@@_qpoint:n
5468             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5469         }
5470         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5471     }
5472 }
5473 {
5474     \pgfpathrectanglecorners
5475     { \@@_qpoint:n { row-1 } }
5476     {
5477         \pgfpointadd
5478         {
5479             \@@_qpoint:n
5480             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5481         }
5482         { \pgfpoint \c_zero_dim \arrayrulewidth }
5483     }
5484 }
5485 \pgfusepath { clip }
5486 \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5487     }
5488 }

```

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`).

```

5489 \cs_new_protected:Npn \@@_actually_color:
5490 {
5491     \pgfpicture
5492     \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.

```

5493     \@@_clip_with_rounded_corners:
5494     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5495     {
5496         \int_compare:nNnTF { ##1 } = \c_one_int
5497         {
5498             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5499             \use:c { g_@@_color _ 1 _tl }
5500             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5501         }
5502         {
5503             \begin { pgfscope }
5504                 \@@_color_opacity ##2
5505                 \use:c { g_@@_color _ ##1 _tl }
5506                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5507                 \pgfusepath { fill }
5508             \end { pgfscope }
5509         }
5510     }
5511 \endpgfpicture
5512 }

```

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

```

5513 \cs_new_protected:Npn \@@_color_opacity
5514 {
5515   \peek_meaning:NTF [
5516     { \@@_color_opacity:w }
5517     { \@@_color_opacity:w [ ] }
5518   }

```

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.

```

5519 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5520 {
5521   \tl_clear:N \l_tmpa_tl
5522   \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.

```

5523   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5524   \tl_if_empty:NTF \l_tmpb_tl
5525     { \@declaredcolor }
5526     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5527 }

```

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

```

5528 \keys_define:nn { nicematrix / color-opacity }
5529 {
5530   opacity .tl_set:N          = \l_tmpa_tl ,
5531   opacity .value_required:n = true
5532 }

```

```

5533 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5534 {
5535   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5536   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5537   \@@_cartesian_path:
5538 }

```

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

```

5539 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5540 {
5541   \tl_if_blank:nF { #2 }
5542   {
5543     \@@_add_to_colors_seq:en
5544     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5545     { \@@_cartesian_color:nn { #3 } { - } }
5546   }
5547 }

```

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

```

5548 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5549 {
5550   \tl_if_blank:nF { #2 }
5551   {
5552     \@@_add_to_colors_seq:en
5553     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5554     { \@@_cartesian_color:nn { - } { #3 } }
5555   }
5556 }

```

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

```

5557 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5558 {
5559   \tl_if_blank:nF { #2 }
5560   {
5561     \@@_add_to_colors_seq:en
5562     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5563     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5564   }
5565 }

```

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

```

5566 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5567 {
5568   \tl_if_blank:nF { #2 }
5569   {
5570     \@@_add_to_colors_seq:en
5571     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5572     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5573   }
5574 }

```

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

```

5575 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5576 {
5577   \@@_cut_on_hyphen:w #1 \q_stop
5578   \tl_clear_new:N \l_@@_tmpc_tl
5579   \tl_clear_new:N \l_@@_tmpd_tl
5580   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5581   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5582   \@@_cut_on_hyphen:w #2 \q_stop
5583   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5584   \tl_set:Nx \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

5585   \@@_cartesian_path:n { #3 }
5586 }

```

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

```

5587 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5588 {
5589   \clist_map_inline:nn { #3 }
5590   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5591 }

```

```

5592 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5593 {
5594   \int_step_inline:nn \c@iRow
5595   {
5596     \int_step_inline:nn \c@jCol
5597     {
5598       \int_if_even:nTF { #####1 + ##1 }
5599       { \@@_cellcolor [ #1 ] { #2 } }
5600       { \@@_cellcolor [ #1 ] { #3 } }
5601       { ##1 - #####1 }
5602     }
5603   }
5604 }

```

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”.

```

5605 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5606 {
5607   \@@_rectanglecolor [ #1 ] { #2 }
5608   { 1 - 1 }
5609   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5610 }

5611 \keys_define:nn { NiceMatrix / rowcolors }
5612 {
5613   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5614   respect-blocks .default:n = true ,
5615   cols .tl_set:N = \l_@@_cols_tl ,
5616   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5617   restart .default:n = true ,
5618   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5619 }

```

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*.

```

5620 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5621 {

```

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

```

5622   \group_begin:
5623   \seq_clear_new:N \l_@@_colors_seq
5624   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5625   \tl_clear_new:N \l_@@_cols_tl
5626   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5627   \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).

```

5628   \int_zero_new:N \l_@@_color_int
5629   \int_set_eq:NN \l_@@_color_int \c_one_int
5630   \bool_if:NT \l_@@_respect_blocks_bool
5631   {

```

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 a the sequence `\l_tmpa_seq`).

```

5632     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5633     \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5634     { \@@_not_in_exterior_p:nnnnn ##1 }
5635   }
5636   \pgfpicture
5637   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5638   \clist_map_inline:nn { #2 }
5639   {
5640     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5641     \tl_if_in:NnTF \l_tmpa_tl { - }
5642     { \@@_cut_on_hyphen:w ##1 \q_stop }
5643     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, `l_tmpa_tl` and `l_tmppb_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.

```

5644 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5645 \int_set:Nn \l_@@_color_int
5646 { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5647 \int_zero_new:N \l_@@_tmpc_int
5648 \int_set:Nn \l_@@_tmpc_int \l_tmppb_tl
5649 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5650 {

```

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

```

5651 \int_set_eq:NN \l_tmppb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

5652 \bool_if:NT \l_@@_respect_blocks_bool
5653 {
5654   \seq_set_filter:NNn \l_tmppb_seq \l_tmpa_seq
5655   { \@@_intersect_our_row_p:nnnnn #####1 }
5656   \seq_map_inline:Nn \l_tmppb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5657 }
5658 \tl_set:No \l_@@_rows_tl
5659 { \int_use:N \l_tmpa_int - \int_use:N \l_tmppb_int }

```

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

```

5660 \tl_clear_new:N \l_@@_color_tl
5661 \tl_set:Nx \l_@@_color_tl
5662 {
5663   \@@_color_index:n
5664   {
5665     \int_mod:nn
5666     { \l_@@_color_int - 1 }
5667     { \seq_count:N \l_@@_colors_seq }
5668     + 1
5669   }
5670 }
5671 \tl_if_empty:NF \l_@@_color_tl
5672 {
5673   \@@_add_to_colors_seq:ee
5674   { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5675   { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5676 }
5677 \int_incr:N \l_@@_color_int
5678 \int_set:Nn \l_tmpa_int { \l_tmppb_int + 1 }
5679 }
5680 }
5681 \endpgfpicture
5682 \group_end:
5683 }

```

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.

```

5684 \cs_new:Npn \@@_color_index:n #1
5685 {
5686   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5687   { \@@_color_index:n { #1 - 1 } }
5688   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5689 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5690 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5691 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5692 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5693 {
5694   \int_compare:nNtT { #3 } > \l_tmpb_int
5695   { \int_set:Nn \l_tmpb_int { #3 } }
5696 }

5697 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5698 {
5699   \int_if_zero:nTF { #4 }
5700   \prg_return_false:
5701   {
5702     \int_compare:nNtTF { #2 } > \c@jCol
5703     \prg_return_false:
5704     \prg_return_true:
5705   }
5706 }

```

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

```

5707 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5708 {
5709   \int_compare:nNtTF { #1 } > \l_tmpa_int
5710   \prg_return_false:
5711   {
5712     \int_compare:nNtTF \l_tmpa_int > { #3 }
5713     \prg_return_false:
5714     \prg_return_true:
5715   }
5716 }

```

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).

```

5717 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5718 {
5719   \dim_compare:nNtTF { #1 } = \c_zero_dim
5720   {
5721     \bool_if:NTF
5722     \@@_nocolor_used_bool
5723     \@@_cartesian_path_normal_ii:
5724     {
5725       \seq_if_empty:NTF \l_@@_corners_cells_seq
5726       { \@@_cartesian_path_normal_i:n { #1 } }
5727       \@@_cartesian_path_normal_ii:
5728     }
5729   }
5730   { \@@_cartesian_path_normal_i:n { #1 } }
5731 }

```

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.

```

5732 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5733 {
5734   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5735 \clist_map_inline:Nn \l_@@_cols_tl
5736 {
5737   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5738   \tl_if_in:NnTF \l_tmpa_tl { - }
5739     { \@@_cut_on_hyphen:w ##1 \q_stop }
5740     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5741   \tl_if_empty:NNTF \l_tmpa_tl
5742     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5743     {
5744       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5745       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5746     }
5747   \tl_if_empty:NNTF \l_tmpb_tl
5748     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5749     {
5750       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5751       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5752     }
5753   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5754     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5755   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5756   \@@_qpoint:n { col - \l_tmpa_tl }
5757   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5758     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5759     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5760   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5761   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5762 \clist_map_inline:Nn \l_@@_rows_tl
5763 {
5764   \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5765   \tl_if_in:NnTF \l_tmpa_tl { - }
5766     { \@@_cut_on_hyphen:w #####1 \q_stop }
5767     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5768   \tl_if_empty:NNTF \l_tmpa_tl
5769     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5770     {
5771       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5772       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5773     }
5774   \tl_if_empty:NNTF \l_tmpb_tl
5775     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5776     {
5777       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5778       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5779     }
5780   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5781     { \tl_set:No \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`.

```

5782   \cs_if_exist:cF
5783     { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5784     {
5785       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5786       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5787       \@@_qpoint:n { row - \l_tmpa_tl }
5788       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5789       \pgfpathrectanglecorners
5790         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5791         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5792     }

```



```

5793     }
5794   }
5795 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5796 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5797 {
5798   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5799   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5800   \clist_map_inline:Nn \l_@@_cols_tl
5801   {
5802     \@@_qpoint:n { col - ##1 }
5803     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5804       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5805       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5806     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5807     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5808   \clist_map_inline:Nn \l_@@_rows_tl
5809   {
5810     \seq_if_in:NnF \l_@@_corners_cells_seq
5811     { #####1 - ##1 }
5812     {
5813       \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5814       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5815       \@@_qpoint:n { row - #####1 }
5816       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5817       \cs_if_exist:cF
5818       { @@ _ #####1 _ ##1 _ nocolor }
5819       {
5820         \pgfpathrectanglecorners
5821         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5822         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5823       }
5824     }
5825   }
5826 }
5827 }

```

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`).

```

5828 \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

```

5829 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5830 {
5831   \bool_set_true:N \@@_nocolor_used_bool
5832   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5833   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5834   \clist_map_inline:Nn \l_@@_rows_tl
5835   {
5836     \clist_map_inline:Nn \l_@@_cols_tl
5837     { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5838   }
5839 }

```

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.

```

5840 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5841 {
5842   \clist_set_eq:NN \l_tmpa_clist #1
5843   \clist_clear:N #1
5844   \clist_map_inline:Nn \l_tmpa_clist
5845   {
5846     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5847     \tl_if_in:NnTF \l_tmpa_tl { - }
5848     { \@@_cut_on_hyphen:w ##1 \q_stop }
5849     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5850     \bool_lazy_or:nnT
5851     { \tl_if_blank_p:o \l_tmpa_tl }
5852     { \str_if_eq_p:on \l_tmpa_tl { * } }
5853     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5854     \bool_lazy_or:nnT
5855     { \tl_if_blank_p:o \l_tmpb_tl }
5856     { \str_if_eq_p:on \l_tmpb_tl { * } }
5857     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5858     \int_compare:nNnT \l_tmpb_tl > #2
5859     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5860     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5861     { \clist_put_right:Nn #1 { ####1 } }
5862   }
5863 }

```

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

```

5864 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5865 {
5866   \@@_test_color_inside:
5867   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5868   {

```

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`).

```

5869     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5870     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5871   }
5872   \ignorespaces
5873 }

```

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

```

5874 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5875 {
5876   \@@_test_color_inside:
5877   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5878   {
5879     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5880     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5881     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5882   }
5883   \ignorespaces
5884 }

```

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).

```

5885 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5886 { \@@_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.

```

5887 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5888 {
5889   \@@_test_color_inside:
5890   \peek_remove_spaces:n
5891   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5892 }

5893 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5894 {

```

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`.

```

5895   \seq_gclear:N \g_tmpa_seq
5896   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5897   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5898   \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`.

```

5899   \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5900   {
5901     { \int_use:N \c@iRow }
5902     { \exp_not:n { #1 } }
5903     { \exp_not:n { #2 } }
5904     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5905   }
5906 }

```

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`).

```

5907 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5908 {
5909   \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).

```

5910   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5911   {
5912     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5913     {
5914       \@@_rowlistcolors
5915       [ \exp_not:n { #2 } ]
5916       { #1 - \int_eval:n { \c@iRow - 1 } }
5917       { \exp_not:n { #3 } }
5918       [ \exp_not:n { #4 } ]
5919     }
5920   }
5921 }

```

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.

```

5922 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5923 {
5924   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5925     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5926   \seq_gclear:N \g_@@_rowlistcolors_seq
5927 }

5928 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5929 {
5930   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5931     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5932 }

```

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.

```

5933 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5934 {

```

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).

```

5935   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5936   {

```

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).

```

5937     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5938     {
5939       \exp_not:N \columncolor [ #1 ]
5940       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5941     }
5942   }
5943 }

5944 \hook_gput_code:nnn { begindocument } { . }
5945 {
5946   \IfPackageLoadedTF { colortbl }
5947   {
5948     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5949     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5950     \cs_new_protected:Npn \@@_revert_colortbl:
5951     {
5952       \hook_gput_code:nnn { env / tabular / begin } { . }
5953       {
5954         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5955         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5956       }
5957     }
5958   }
5959   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5960 }

```

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`.

```
5961 \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`.

```
5962 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5963 {
5964   \int_if_zero:nTF \l_@@_first_col_int
5965     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5966     {
5967       \int_if_zero:nTF \c@jCol
5968       {
5969         \int_compare:nNnF \c@iRow = { -1 }
5970         { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5971       }
5972       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5973     }
5974 }
```

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.

```
5975 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5976 {
5977   \int_if_zero:nF \c@iRow
5978   {
5979     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5980     {
5981       \int_compare:nNnT \c@jCol > \c_zero_int
5982       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5983     }
5984   }
5985 }
```

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.

```
5986 \keys_define:nn { NiceMatrix / Rules }
5987 {
5988   position .int_set:N = \l_@@_position_int ,
5989   position .value_required:n = true ,
5990   start .int_set:N = \l_@@_start_int ,
```

```

5991     end .code:n =
5992         \bool_lazy_or:nnTF
5993         { \tl_if_empty_p:n { #1 } }
5994         { \str_if_eq_p:nn { #1 } { last } }
5995         { \int_set_eq:NN \l_@@_end_int \c@jCol }
5996         { \int_set:Nn \l_@@_end_int { #1 } }
5997     }

```

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.

```

5998 \keys_define:nn { NiceMatrix / RulesBis }
5999 {
6000     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6001     multiplicity .initial:n = 1 ,
6002     dotted .bool_set:N = \l_@@_dotted_bool ,
6003     dotted .initial:n = false ,
6004     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]`.

```

6005     color .code:n =
6006         \@@_set_CT@arc@:n { #1 }
6007         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6008     color .value_required:n = true ,
6009     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6010     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.

```

6011     tikz .code:n =
6012         \IfPackageLoadedTF { tikz }
6013         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6014         { \@@_error:n { tikz-without-tikz } } ,
6015     tikz .value_required:n = true ,
6016     total-width .dim_set:N = \l_@@_rule_width_dim ,
6017     total-width .value_required:n = true ,
6018     width .meta:n = { total-width = #1 } ,
6019     unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
6020 }

```

The vertical rules

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

```

6021 \cs_new_protected:Npn \@@_vline:n #1
6022 {

```

The group is for the options.

```

6023     \group_begin:
6024     \int_set_eq:NN \l_@@_end_int \c@iRow
6025     \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).

```

6026     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6027     \@@_vline_i:
6028     \group_end:
6029 }

```

```

6030 \cs_new_protected:Npn \@@_vline_i:
6031 {

```

\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.

```

6032 \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6033 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6034 \l_tmpa_tl
6035 {

```

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.

```

6036 \bool_gset_true:N \g_tmpa_bool
6037 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6038 { \@@_test_vline_in_block:nnnnn ##1 }
6039 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6040 { \@@_test_vline_in_block:nnnnn ##1 }
6041 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6042 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6043 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6044 \bool_if:NTF \g_tmpa_bool
6045 {
6046 \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.

```

6047 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6048 }
6049 {
6050 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6051 {
6052 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6053 \@@_vline_ii:
6054 \int_zero:N \l_@@_local_start_int
6055 }
6056 }
6057 }
6058 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6059 {
6060 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6061 \@@_vline_ii:
6062 }
6063 }

```

```

6064 \cs_new_protected:Npn \@@_test_in_corner_v:
6065 {
6066 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6067 {
6068 \seq_if_in:NxT
6069 \l_@@_corners_cells_seq
6070 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6071 { \bool_set_false:N \g_tmpa_bool }
6072 }
6073 {
6074 \seq_if_in:NxT
6075 \l_@@_corners_cells_seq
6076 { \l_tmpa_tl - \l_tmpb_tl }
6077 {
6078 \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6079 { \bool_set_false:N \g_tmpa_bool }
6080 {

```

```

6081         \seq_if_in:NxT
6082         \l_@@_corners_cells_seq
6083         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6084         { \bool_set_false:N \g_tmpa_bool }
6085     }
6086 }
6087 }
6088 }

6089 \cs_new_protected:Npn \@@_vline_ii:
6090 {
6091     \tl_clear:N \l_@@_tikz_rule_tl
6092     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6093     \bool_if:NTF \l_@@_dotted_bool
6094     \@@_vline_iv:
6095     {
6096         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6097         \@@_vline_iii:
6098         \@@_vline_v:
6099     }
6100 }

```

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

```

6101 \cs_new_protected:Npn \@@_vline_iii:
6102 {
6103     \pgfpicture
6104     \pgfrememberpicturepositiononpagetrue
6105     \pgf@relevantforpicturesizefalse
6106     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6107     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6108     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6109     \dim_set:Nn \l_tmpb_dim
6110     {
6111         \pgf@x
6112         - 0.5 \l_@@_rule_width_dim
6113         +
6114         ( \arrayrulewidth * \l_@@_multiplicity_int
6115           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6116     }
6117     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6118     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6119     \bool_lazy_all:nT
6120     {
6121         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6122         { \cs_if_exist_p:N \CT@drsc@ }
6123         { ! \tl_if_blank_p:o \CT@drsc@ }
6124     }
6125     {
6126         \group_begin:
6127         \CT@drsc@
6128         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6129         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6130         \dim_set:Nn \l_@@_tmpd_dim
6131         {
6132             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6133             * ( \l_@@_multiplicity_int - 1 )
6134         }
6135         \pgfpathrectanglecorners
6136         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6137         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6138         \pgfusepath { fill }
6139         \group_end:

```



```

6140     }
6141     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6142     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6143     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6144     {
6145         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6146         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6147         \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6148         \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6149     }
6150     \CT@arc@
6151     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6152     \pgfsetrectcap
6153     \pgfusepathqstroke
6154     \endpgfpicture
6155 }

```

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

```

6156 \cs_new_protected:Npn \@@_vline_iv:
6157 {
6158     \pgfpicture
6159     \pgfrememberpicturepositiononpagetrue
6160     \pgf@relevantforpicturesizefalse
6161     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6162     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6163     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6164     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6165     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6166     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6167     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6168     \CT@arc@
6169     \@@_draw_line:
6170     \endpgfpicture
6171 }

```

The following code is for the case when the user uses the key `tikz`.

```

6172 \cs_new_protected:Npn \@@_vline_v:
6173 {
6174     \begin {tikzpicture }
6175     % added 2023/09/25

```

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).

```

6176     \CT@arc@
6177     \tl_if_empty:NF \l_@@_rule_color_tl
6178     { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6179     \pgfrememberpicturepositiononpagetrue
6180     \pgf@relevantforpicturesizefalse
6181     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6182     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6183     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6184     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6185     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6186     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6187     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6188     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6189     ( \l_tmpb_dim , \l_tmpa_dim ) --
6190     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6191     \end {tikzpicture }
6192 }

```

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).

```

6193 \cs_new_protected:Npn \@@_draw_vlines:
6194 {
6195   \int_step_inline:nnn
6196     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6197     {
6198       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6199       \c@jCol
6200       { \int_eval:n { \c@jCol + 1 } }
6201     }
6202     {
6203       \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6204       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6205       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6206     }
6207 }

```

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}`.

```

6208 \cs_new_protected:Npn \@@_hline:n #1
6209 {

```

The group is for the options.

```

6210   \group_begin:
6211   \int_zero_new:N \l_@@_end_int
6212   \int_set_eq:NN \l_@@_end_int \c@jCol
6213   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6214   \@@_hline_i:
6215   \group_end:
6216 }
6217 \cs_new_protected:Npn \@@_hline_i:
6218 {
6219   \int_zero_new:N \l_@@_local_start_int
6220   \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`.

```

6221   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6222   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6223     \l_tmpb_tl
6224     {

```

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.

```

6225       \bool_gset_true:N \g_tmpa_bool
6226       \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6227         { \@@_test_hline_in_block:nnnnn ##1 }
6228       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6229         { \@@_test_hline_in_block:nnnnn ##1 }
6230       \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6231         { \@@_test_hline_in_stroken_block:nnnn ##1 }
6232       \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6233       \bool_if:NTF \g_tmpa_bool
6234         {
6235           \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.

```

6236         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6237     }
6238     {
6239         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6240         {
6241             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6242             \@@_hline_ii:
6243             \int_zero:N \l_@@_local_start_int
6244         }
6245     }
6246 }
6247 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6248 {
6249     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6250     \@@_hline_ii:
6251 }
6252 }
```

```

6253 \cs_new_protected:Npn \@@_test_in_corner_h:
6254 {
6255     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6256     {
6257         \seq_if_in:NxT
6258         \l_@@_corners_cells_seq
6259         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6260         { \bool_set_false:N \g_tmpa_bool }
6261     }
6262     {
6263         \seq_if_in:NxT
6264         \l_@@_corners_cells_seq
6265         { \l_tmpa_tl - \l_tmpb_tl }
6266         {
6267             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6268             { \bool_set_false:N \g_tmpa_bool }
6269             {
6270                 \seq_if_in:NxT
6271                 \l_@@_corners_cells_seq
6272                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6273                 { \bool_set_false:N \g_tmpa_bool }
6274             }
6275         }
6276     }
6277 }
```

```

6278 \cs_new_protected:Npn \@@_hline_ii:
6279 {
6280     \tl_clear:N \l_@@_tikz_rule_tl
6281     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6282     \bool_if:NTF \l_@@_dotted_bool
6283     \@@_hline_iv:
6284     {
6285         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6286         \@@_hline_iii:
6287         \@@_hline_v:
6288     }
6289 }
```

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

```

6290 \cs_new_protected:Npn \@@_hline_iii:
```

```

6291 {
6292   \pgfpicture
6293   \pgfrememberpicturepositiononpagetrue
6294   \pgf@relevantforpicturesizefalse
6295   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6296   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6297   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6298   \dim_set:Nn \l_tmpb_dim
6299     {
6300       \pgf@y
6301       - 0.5 \l_@@_rule_width_dim
6302       +
6303       ( \arrayrulewidth * \l_@@_multiplicity_int
6304         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6305     }
6306   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6307   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6308   \bool_lazy_all:nT
6309     {
6310       { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6311       { \cs_if_exist_p:N \CT@drsc@ }
6312       { ! \tl_if_blank_p:o \CT@drsc@ }
6313     }
6314     {
6315       \group_begin:
6316       \CT@drsc@
6317       \dim_set:Nn \l_@@_tmpd_dim
6318         {
6319           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6320           * ( \l_@@_multiplicity_int - 1 )
6321         }
6322       \pgfpathrectanglecorners
6323         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6324         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6325       \pgfusepathqfill
6326       \group_end:
6327     }
6328   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6329   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6330   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6331     {
6332       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6333       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6334       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6335       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6336     }
6337   \CT@arc@
6338   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6339   \pgfsetrectcap
6340   \pgfusepathqstroke
6341   \endpgfpicture
6342 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

```

6343 \cs_new_protected:Npn \@@_hline_iv:
6344 {
6345   \pgfpicture
6346   \pgfrememberpicturepositiononpagetrue
6347   \pgf@relevantforpicturesizefalse
6348   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6349   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6350   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6351   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6352   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6353   \int_compare:nNtT \l_@@_local_start_int = \c_one_int
6354   {
6355     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6356     \bool_if:NF \g_@@_delims_bool
6357     { \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.

```

6358   \tl_if_eq:NnF \g_@@_left_delim_tl (
6359     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6360   )
6361   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6362   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6363   \int_compare:nNtT \l_@@_local_end_int = \c_jCol
6364   {
6365     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6366     \bool_if:NF \g_@@_delims_bool
6367     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6368     \tl_if_eq:NnF \g_@@_right_delim_tl )
6369     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6370   }
6371   \CT@arc@
6372   \@@_draw_line:
6373   \endpgfpicture
6374 }

```

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`).

```

6375 \cs_new_protected:Npn \@@_hline_v:
6376 {
6377   \begin { tikzpicture }
6378   % added 2023/09/25

```

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).

```

6379   \CT@arc@
6380   \tl_if_empty:NF \l_@@_rule_color_tl
6381   { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6382   \pgfrememberpicturepositiononpagetrue
6383   \pgf@relevantforpicturesizefalse
6384   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6385   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6386   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6387   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6388   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }

```

```

6389 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6390 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6391 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6392 ( \l_tmpa_dim , \l_tmpb_dim ) --
6393 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6394 \end { tikzpicture }
6395 }

```

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).

```

6396 \cs_new_protected:Npn \@@_draw_hlines:
6397 {
6398   \int_step_inline:nnn
6399   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6400   {
6401     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6402     \c@iRow
6403     { \int_eval:n { \c@iRow + 1 } }
6404   }
6405   {
6406     \tl_if_eq:NnF \l_@@_hlines_clist \c_@@_all_tl
6407     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6408     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6409   }
6410 }

```

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

```

6411 \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.

```

6412 \cs_set:Npn \@@_Hline_i:n #1
6413 {
6414   \peek_remove_spaces:n
6415   {
6416     \peek_meaning:NNTF \Hline
6417     { \@@_Hline_ii:nn { #1 + 1 } }
6418     { \@@_Hline_iii:n { #1 } }
6419   }
6420 }
6421 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6422 \cs_set:Npn \@@_Hline_iii:n #1
6423 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6424 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6425 {
6426   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6427   \skip_vertical:N \l_@@_rule_width_dim
6428   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6429   {
6430     \@@_hline:n
6431     {
6432       multiplicity = #1 ,
6433       position = \int_eval:n { \c@iRow + 1 } ,
6434       total-width = \dim_use:N \l_@@_rule_width_dim ,
6435       #2
6436     }
6437   }
6438   \egroup
6439 }

```

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`).

```

6440 \cs_new_protected:Npn \@@_custom_line:n #1
6441 {
6442   \str_clear_new:N \l_@@_command_str
6443   \str_clear_new:N \l_@@_ccommand_str
6444   \str_clear_new:N \l_@@_letter_str
6445   \tl_clear_new:N \l_@@_other_keys_tl
6446   \keys_set:known:nN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6447   \bool_lazy_all:nTF
6448   {
6449     { \str_if_empty_p:N \l_@@_letter_str }
6450     { \str_if_empty_p:N \l_@@_command_str }
6451     { \str_if_empty_p:N \l_@@_ccommand_str }
6452   }
6453   { \@@_error:n { No~letter-and-no~command } }
6454   { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6455 }

6456 \keys_define:nn { NiceMatrix / custom-line }
6457 {
6458   letter .str_set:N = \l_@@_letter_str ,
6459   letter .value_required:n = true ,
6460   command .str_set:N = \l_@@_command_str ,
6461   command .value_required:n = true ,
6462   ccommand .str_set:N = \l_@@_ccommand_str ,
6463   ccommand .value_required:n = true ,
6464 }

```

```

6465 \cs_new_protected:Npn \@@_custom_line_i:n #1
6466 {

```

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

```

6467   \bool_set_false:N \l_@@_tikz_rule_bool
6468   \bool_set_false:N \l_@@_dotted_rule_bool
6469   \bool_set_false:N \l_@@_color_bool

6470   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6471   \bool_if:NT \l_@@_tikz_rule_bool
6472   {
6473     \IfPackageLoadedTF { tikz }
6474     { }
6475     { \@@_error:n { tikz~in~custom-line~without~tikz } }
6476     \bool_if:NT \l_@@_color_bool
6477     { \@@_error:n { color~in~custom-line~with~tikz } }
6478   }
6479   \bool_if:NT \l_@@_dotted_rule_bool
6480   {
6481     \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6482     { \@@_error:n { key~multiplicity~with~dotted } }
6483   }
6484   \str_if_empty:NF \l_@@_letter_str
6485   {

```

```

6486 \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6487 { \@@_error:n { Several~letters } }
6488 {
6489 \exp_args:NnV \tl_if_in:NnTF
6490 \c_@@_forbidden_letters_str \l_@@_letter_str
6491 { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6492 {

```

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.

```

6493 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6494 { \@@_v_custom_line:n { #1 } }
6495 }
6496 }
6497 }
6498 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6499 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6500 }
6501 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6502 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6503 \keys_define:nn { NiceMatrix / custom-line-bis }
6504 {
6505 multiplicity .int_set:N = \l_@@_multiplicity_int ,
6506 multiplicity .initial:n = 1 ,
6507 multiplicity .value_required:n = true ,
6508 color .code:n = \bool_set_true:N \l_@@_color_bool ,
6509 color .value_required:n = true ,
6510 tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6511 tikz .value_required:n = true ,
6512 dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6513 dotted .value_forbidden:n = true ,
6514 total-width .code:n = { } ,
6515 total-width .value_required:n = true ,
6516 width .code:n = { } ,
6517 width .value_required:n = true ,
6518 sep-color .code:n = { } ,
6519 sep-color .value_required:n = true ,
6520 unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6521 }

```

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

```

6522 \bool_new:N \l_@@_dotted_rule_bool
6523 \bool_new:N \l_@@_tikz_rule_bool
6524 \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`.

```

6525 \keys_define:nn { NiceMatrix / custom-line-width }
6526 {
6527 multiplicity .int_set:N = \l_@@_multiplicity_int ,
6528 multiplicity .initial:n = 1 ,
6529 multiplicity .value_required:n = true ,
6530 tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6531 total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6532 \bool_set_true:N \l_@@_total_width_bool ,

```



```

6533     total-width .value_required:n = true ,
6534     width .meta:n = { total-width = #1 } ,
6535     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6536 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. #1 is the whole set of keys to pass to the command \@@_hline:n (which is in the internal \CodeAfter).

```

6537 \cs_new_protected:Npn \@@_h_custom_line:n #1
6538 {

```

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).

```

6539     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6540     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6541 }

```

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).

```

6542 \cs_new_protected:Npn \@@_c_custom_line:n #1
6543 {

```

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

```

6544     \exp_args:Nc \NewExpandableDocumentCommand
6545     { nicematrix - \l_@@_ccommand_str }
6546     { 0 { } m }
6547     {
6548         \noalign
6549         {
6550             \@@_compute_rule_width:n { #1 , ##1 }
6551             \skip_vertical:n { \l_@@_rule_width_dim }
6552             \clist_map_inline:nn
6553             { ##2 }
6554             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6555         }
6556     }
6557     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6558 }

```

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*.

```

6559 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6560 {
6561     \str_if_in:nnTF { #2 } { - }
6562     { \@@_cut_on_hyphen:w #2 \q_stop }
6563     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6564     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6565     {
6566         \@@_hline:n
6567         {
6568             #1 ,
6569             start = \l_tmpa_tl ,
6570             end = \l_tmpb_tl ,
6571             position = \int_eval:n { \c@iRow + 1 } ,
6572             total-width = \dim_use:N \l_@@_rule_width_dim
6573         }
6574     }
6575 }

```

```

6576 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6577 {
6578   \bool_set_false:N \l_@@_tikz_rule_bool
6579   \bool_set_false:N \l_@@_total_width_bool
6580   \bool_set_false:N \l_@@_dotted_rule_bool
6581   \keys_set_known:n { NiceMatrix / custom-line-width } { #1 }
6582   \bool_if:NF \l_@@_total_width_bool
6583   {
6584     \bool_if:NTF \l_@@_dotted_rule_bool
6585     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6586     {
6587       \bool_if:NF \l_@@_tikz_rule_bool
6588       {
6589         \dim_set:Nn \l_@@_rule_width_dim
6590         {
6591           \arrayrulewidth * \l_@@_multiplicity_int
6592           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6593         }
6594       }
6595     }
6596   }
6597 }
6598 \cs_new_protected:Npn \@@_v_custom_line:n #1
6599 {
6600   \@@_compute_rule_width:n { #1 }

```

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

```

6601   \tl_gput_right:Nx \g_@@_array_preamble_tl
6602   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6603   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6604   {
6605     \@@_vline:n
6606     {
6607       #1 ,
6608       position = \int_eval:n { \c@jCol + 1 } ,
6609       total-width = \dim_use:N \l_@@_rule_width_dim
6610     }
6611   }
6612   \@@_rec_preamble:n
6613 }
6614 \@@_custom_line:n
6615 { 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.

```

6616 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6617 {
6618   \int_compare:nNnT \l_tmpa_tl > { #1 }
6619   {
6620     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6621     {
6622       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6623       {
6624         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6625         { \bool_gset_false:N \g_tmpa_bool }
6626       }
6627     }
6628   }
6629 }

```

The same for vertical rules.

```

6630 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6631 {
6632   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6633   {
6634     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6635     {
6636       \int_compare:nNnT \l_tmpb_tl > { #2 }
6637       {
6638         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6639         { \bool_gset_false:N \g_tmpa_bool }
6640       }
6641     }
6642   }
6643 }

6644 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6645 {
6646   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6647   {
6648     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6649     {
6650       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6651       { \bool_gset_false:N \g_tmpa_bool }
6652       {
6653         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6654         { \bool_gset_false:N \g_tmpa_bool }
6655       }
6656     }
6657   }
6658 }

6659 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6660 {
6661   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6662   {
6663     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6664     {
6665       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6666       { \bool_gset_false:N \g_tmpa_bool }
6667       {
6668         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6669         { \bool_gset_false:N \g_tmpa_bool }
6670       }
6671     }
6672   }
6673 }

```

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.

```

6674 \cs_new_protected:Npn \@@_compute_corners:
6675 {

```

The sequence `\l_@@_corners_cells_seq` will be the sequence of all the empty cells (and not in a block) considered in the corners of the array.

```

6676   \seq_clear_new:N \l_@@_corners_cells_seq

```

```

6677 \clist_map_inline:Nn \l_@@_corners_clist
6678 {
6679   \str_case:nnF { ##1 }
6680   {
6681     { NW }
6682     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6683     { NE }
6684     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6685     { SW }
6686     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6687     { SE }
6688     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6689   }
6690   { \@@_error:nn { bad~corner } { ##1 } }
6691 }

```

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

```

6692 \seq_if_empty:NF \l_@@_corners_cells_seq
6693 {

```

You write on the aux file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which color the `rows`, `columns` and `cells` must not color the cells in the corners.

```

6694 \tl_build_gput_right:Nx \g_@@_aux_tl
6695 {
6696   \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6697   { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6698 }
6699 }
6700 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_seq`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- `#1` and `#2` are the number of row and column of the cell which is actually in the corner;
- `#3` and `#4` are the steps in rows and the step in columns when moving from the corner;
- `#5` is the number of the final row when scanning the rows from the corner;
- `#6` is the number of the final column when scanning the columns from the corner.

```

6701 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6702 {

```

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.

```

6703 \bool_set_false:N \l_tmpa_bool
6704 \int_zero_new:N \l_@@_last_empty_row_int
6705 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6706 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6707 {
6708   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6709   \bool_lazy_or:nnTF
6710   {
6711     \cs_if_exist_p:c
6712     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6713   }
6714   \l_tmpb_bool
6715   { \bool_set_true:N \l_tmpa_bool }

```

```

6716     {
6717         \bool_if:NF \l_tmpa_bool
6718         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6719     }
6720 }

```

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

```

6721 \bool_set_false:N \l_tmpa_bool
6722 \int_zero_new:N \l_@@_last_empty_column_int
6723 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6724 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6725 {
6726     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6727     \bool_lazy_or:nnTF
6728         \l_tmpb_bool
6729         {
6730             \cs_if_exist_p:c
6731             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6732         }
6733         { \bool_set_true:N \l_tmpa_bool }
6734         {
6735             \bool_if:NF \l_tmpa_bool
6736             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6737         }
6738     }

```

Now, we loop over the rows.

```

6739 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6740 {

```

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

```

6741     \bool_set_false:N \l_tmpa_bool
6742     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6743     {
6744         \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6745         \bool_lazy_or:nnTF
6746             \l_tmpb_bool
6747             {
6748                 \cs_if_exist_p:c
6749                 { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6750             }
6751             { \bool_set_true:N \l_tmpa_bool }
6752             {
6753                 \bool_if:NF \l_tmpa_bool
6754                 {
6755                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6756                     \seq_put_right:Nn
6757                         \l_@@_corners_cells_seq
6758                         { ##1 - #####1 }
6759                 }
6760             }
6761     }
6762 }
6763 }

```

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`).

```

6764 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6765 {
6766     \int_set:Nn \l_tmpa_int { #1 }
6767     \int_set:Nn \l_tmpb_int { #2 }
6768     \bool_set_false:N \l_tmpb_bool
6769     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

6770     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6771   }
6772 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6773 {
6774   \int_compare:nNnF { #3 } > { #1 }
6775   {
6776     \int_compare:nNnF { #1 } > { #5 }
6777     {
6778       \int_compare:nNnF { #4 } > { #2 }
6779       {
6780         \int_compare:nNnF { #2 } > { #6 }
6781         { \bool_set_true:N \l_tmpb_bool }
6782       }
6783     }
6784   }
6785 }

```

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.

```

6786 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6787 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6788 {
6789   auto-columns-width .code:n =
6790   {
6791     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6792     \dim_gzero_new:N \g_@@_max_cell_width_dim
6793     \bool_set_true:N \l_@@_auto_columns_width_bool
6794   }
6795 }

6796 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6797 {
6798   \int_gincr:N \g_@@_NiceMatrixBlock_int
6799   \dim_zero:N \l_@@_columns_width_dim
6800   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6801   \bool_if:NT \l_@@_block_auto_columns_width_bool
6802   {
6803     \cs_if_exist:cT
6804     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6805     {
6806       % is \exp_args:NNe mandatory?
6807       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6808       {
6809         \use:c
6810         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6811       }
6812     }
6813   }
6814 }

```

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).

```

6815 {
6816   \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.

```

6817   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6818   {
6819     \bool_if:NT \l_@@_block_auto_columns_width_bool
6820     {
6821       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6822       \iow_shipout:Nx \@mainaux
6823       {
6824         \cs_gset:cpn
6825         { @@ _ 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.

```

6826         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6827       }
6828       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6829     }
6830   }
6831   \ignorespacesafterend
6832 }

```

26 The extra nodes

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

```

6833 \cs_generate_variant:Nn \dim_min:nn { v n }
6834 \cs_generate_variant:Nn \dim_max:nn { v n }

```

The following command is called in `\@@_use_arraybox_with_notes_c:` just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

```

6835 \cs_new_protected:Npn \@@_create_extra_nodes:
6836 {
6837   \bool_if:nTF \l_@@_medium_nodes_bool
6838   {
6839     \bool_if:NTF \l_@@_large_nodes_bool
6840     \@@_create_medium_and_large_nodes:
6841     \@@_create_medium_nodes:
6842   }
6843   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6844 }

```

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`.

```

6845 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6846 {
6847   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6848   {
6849     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6850     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6851     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6852     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6853   }
6854   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6855   {
6856     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6857     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6858     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6859     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6860   }

```

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

```

6861   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6862   {
6863     \int_step_variable:nnNn
6864     \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.

```

6865     {
6866       \cs_if_exist:cT
6867       { 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`.

```

6868     {
6869       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6870       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6871       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6872       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6873       {
6874         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6875         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6876       }

```

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`.

```

6877       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6878       \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6879       { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6880       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6881       {
6882         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6883         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6884       }
6885     }
6886   }
6887 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6888   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6889   {
6890     \dim_compare:nNnT
6891     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim

```



```

6892     {
6893         \@@_qpoint:n { row - \@@_i: - base }
6894         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6895         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6896     }
6897 }
6898 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6899 {
6900     \dim_compare:nNnT
6901     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6902     {
6903         \@@_qpoint:n { col - \@@_j: }
6904         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6905         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6906     }
6907 }
6908 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6909 \cs_new_protected:Npn \@@_create_medium_nodes:
6910 {
6911     \pgfpicture
6912     \pgfrememberpicturepositiononpagetrue
6913     \pgf@relevantforpicturesizefalse
6914     \@@_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”.

```

6915     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6916     \@@_create_nodes:
6917     \endpgfpicture
6918 }

```

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:`.

```

6919 \cs_new_protected:Npn \@@_create_large_nodes:
6920 {
6921     \pgfpicture
6922     \pgfrememberpicturepositiononpagetrue
6923     \pgf@relevantforpicturesizefalse
6924     \@@_computations_for_medium_nodes:
6925     \@@_computations_for_large_nodes:
6926     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6927     \@@_create_nodes:
6928     \endpgfpicture
6929 }
6930 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6931 {
6932     \pgfpicture
6933     \pgfrememberpicturepositiononpagetrue
6934     \pgf@relevantforpicturesizefalse
6935     \@@_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”.

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6936 \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6937 @@_create_nodes:
6938 @@_computations_for_large_nodes:
6939 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6940 @@_create_nodes:
6941 \endpgfpicture
6942 }

```

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.

```

6943 \cs_new_protected:Npn @@_computations_for_large_nodes:
6944 {
6945 \int_set_eq:NN \l_@@_first_row_int \c_one_int
6946 \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`.

```

6947 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6948 {
6949 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6950 {
6951 (
6952 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6953 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6954 )
6955 / 2
6956 }
6957 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6958 { l_@@_row _ \@@_i: _ min _ dim }
6959 }
6960 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6961 {
6962 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6963 {
6964 (
6965 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6966 \dim_use:c
6967 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6968 )
6969 / 2
6970 }
6971 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6972 { l_@@_column _ \@@_j: _ max _ dim }
6973 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6974 \dim_sub:cn
6975 { l_@@_column _ 1 _ min _ dim }
6976 \l_@@_left_margin_dim
6977 \dim_add:cn
6978 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6979 \l_@@_right_margin_dim
6980 }

```

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`).

```

6981 \cs_new_protected:Npn @@_create_nodes:
6982 {

```

```

6983 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6984 {
6985     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6986     {

```

We draw the rectangular node for the cell ($\backslash\@@_i-\backslash\@@_j$).

```

6987         \@@_pgf_rect_node:nnnnn
6988         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6989         { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6990         { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6991         { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6992         { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6993         \str_if_empty:NF \l_@@_name_str
6994         {
6995             \pgfnodealias
6996             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6997             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6998         }
6999     }
7000 }

```

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 .

```

7001     \seq_map_pairwise_function:NNN
7002     \g_@@_multicolumn_cells_seq
7003     \g_@@_multicolumn_sizes_seq
7004     \@@_node_for_multicolumn:nn
7005 }

7006 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7007 {
7008     \cs_set_nopar:Npn \@@_i: { #1 }
7009     \cs_set_nopar:Npn \@@_j: { #2 }
7010 }

```

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”).

```

7011 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7012 {
7013     \@@_extract_coords_values: #1 \q_stop
7014     \@@_pgf_rect_node:nnnnn
7015     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7016     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7017     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7018     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
7019     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7020     \str_if_empty:NF \l_@@_name_str
7021     {
7022         \pgfnodealias
7023         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7024         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7025     }
7026 }

```

27 The blocks

The code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command \Block will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass.

```

7027 \keys_define:nn { NiceMatrix / Block / FirstPass }
7028 {
7029   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7030   l .value_forbidden:n = true ,
7031   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7032   r .value_forbidden:n = true ,
7033   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7034   c .value_forbidden:n = true ,
7035   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7036   L .value_forbidden:n = true ,
7037   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7038   R .value_forbidden:n = true ,
7039   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7040   C .value_forbidden:n = true ,
7041   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7042   t .value_forbidden:n = true ,
7043   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7044   T .value_forbidden:n = true ,
7045   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7046   b .value_forbidden:n = true ,
7047   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7048   B .value_forbidden:n = true ,
7049   color .code:n =
7050     \@@_color:n { #1 }
7051     \tl_set_rescan:Nnn
7052       \l_@@_draw_tl
7053       { \char_set_catcode_other:N ! }
7054       { #1 } ,
7055   color .value_required:n = true ,
7056   respect-arraystretch .code:n =
7057     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7058   respect-arraystretch .value_forbidden:n = true ,
7059 }

```

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.

```

7060 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7061 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7062 {

```

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).

```

7063   \peek_remove_spaces:n
7064   {
7065     \tl_if_blank:nTF { #2 }
7066     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7067     {
7068       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7069       \@@_Block_i_czech \@@_Block_i
7070       #2 \q_stop
7071     }
7072     { #1 } { #3 } { #4 }
7073   }
7074 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7075 \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`.

```

7076 {
7077   \char_set_catcode_active:N -
7078   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7079 }

```

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.

```

7080 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7081 {

```

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).

```

7082   \bool_lazy_or:nnTF
7083     { \tl_if_blank_p:n { #1 } }
7084     { \str_if_eq_p:nn { #1 } { * } }
7085     { \int_set:Nn \l_tmpa_int { 100 } }
7086     { \int_set:Nn \l_tmpa_int { #1 } }
7087   \bool_lazy_or:nnTF
7088     { \tl_if_blank_p:n { #2 } }
7089     { \str_if_eq_p:nn { #2 } { * } }
7090     { \int_set:Nn \l_tmpb_int { 100 } }
7091     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7092   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7093   {
7094     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7095     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7096     { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7097   }
7098   { \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.

```

7099   \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
7100   \tl_set:Nx \l_tmpa_tl
7101   {
7102     { \int_use:N \c_iRow }
7103     { \int_use:N \c_jCol }
7104     { \int_eval:n { \c_iRow + \l_tmpa_int - 1 } }
7105     { \int_eval:n { \c_jCol + \l_tmpb_int - 1 } }
7106   }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

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

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by currying).

```

7107   \bool_if:nTF
7108   {
7109     (
7110       \int_compare_p:nNn \l_tmpa_int = \c_one_int
7111       ||

```

```

7112     \int_compare_p:nNn \l_tmpb_int = \c_one_int
7113   )
7114   && ! \tl_if_empty_p:n { #5 }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7115     && ! \l_@@_X_bool
7116   }
7117   { \exp_args:Nee \@@_Block_iv:nnnnn }
7118   { \exp_args:Nee \@@_Block_v:nnnnn }
7119   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7120 }

```

The following macro is for the case of a \Block which is mono-row or mono-column (or both). In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) 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.

```

7121 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7122 {
7123   \int_gincr:N \g_@@_block_box_int
7124   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7125   {
7126     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7127     {
7128       \@@_actually_diagbox:nnnnnn
7129       { \int_use:N \c_iRow }
7130       { \int_use:N \c_jCol }
7131       { \int_eval:n { \c_iRow + #1 - 1 } }
7132       { \int_eval:n { \c_jCol + #2 - 1 } }
7133       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7134       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7135     }
7136   }
7137   \box_gclear_new:c
7138   { g_@@_block_box _ \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!

```

7139   \hbox_gset:cn
7140   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
7141   {

```

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}).

```

7142     \tl_if_empty:NTF \l_@@_color_tl
7143     { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7144     { \@@_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.

```

7145 \int_compare:nNnT { #1 } = \c_one_int
7146 {
7147   \int_if_zero:nTF \c@iRow
7148   \l_@@_code_for_first_row_tl
7149   {
7150     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7151     \l_@@_code_for_last_row_tl
7152   }
7153   \g_@@_row_style_tl
7154 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7155 \@@_reset_arraystretch:
7156 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7157 #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.).

```

7158 \@@_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}`.

```

7159 \bool_if:NTF \l_@@_tabular_bool
7160 {
7161   \bool_lazy_all:nTF
7162   {
7163     { \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.

```

7164 { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7165 { ! \g_@@_rotate_bool }
7166 }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7167 {
7168   \use:e
7169   {
7170     \exp_not:N \begin { minipage }%
7171     [ \str_lowercase:V \l_@@_vpos_block_str ]
7172     { \l_@@_col_width_dim }
7173     \str_case:on \l_@@_hpos_block_str
7174     { c \centering r \raggedleft l \raggedright }
7175   }
7176   #5
7177   \end { minipage }
7178 }

```

In the other cases, we use a `{tabular}`.

```

7179 {
7180   \use:e
7181   {
7182     \exp_not:N \begin { tabular }%
7183     [ \str_lowercase:V \l_@@_vpos_block_str ]
7184     { @ { } \l_@@_hpos_block_str @ { } }
7185   }
7186   #5
7187   \end { tabular }
7188 }
7189 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7190     {
7191         \c_math_toggle_token
7192         \use:e
7193         {
7194             \exp_not:N \begin { array }%
7195             [ \str_lowercase:V \l_@@_vpos_block_str ]
7196             { @ { } \l_@@_hpos_block_str @ { } }
7197         }
7198         #5
7199         \end { array }
7200         \c_math_toggle_token
7201     }
7202 }

```

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 the rotated box).

```

7203     \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.

```

7204     \int_compare:nNnT { #2 } = \c_one_int
7205     {
7206         \dim_gset:Nn \g_@@_blocks_wd_dim
7207         {
7208             \dim_max:nn
7209             \g_@@_blocks_wd_dim
7210             {
7211                 \box_wd:c
7212                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7213             }
7214         }
7215     }

```

If we are in a mono-row block and if that block has no vertical option for the position¹⁵, we take into account the height and the depth of that block for the height and the depth of the row.

```

7216     \str_if_eq:VnT \l_@@_vpos_block_str { c }
7217     {
7218         \int_compare:nNnT { #1 } = \c_one_int
7219         {
7220             \dim_gset:Nn \g_@@_blocks_ht_dim
7221             {
7222                 \dim_max:nn
7223                 \g_@@_blocks_ht_dim
7224                 {
7225                     \box_ht:c
7226                     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7227                 }
7228             }
7229             \dim_gset:Nn \g_@@_blocks_dp_dim
7230             {
7231                 \dim_max:nn
7232                 \g_@@_blocks_dp_dim
7233                 {
7234                     \box_dp:c
7235                     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

¹⁵If the block has a key of a vertical position, that means that it has to be put in a vertical space determined by the *others* cells of the row. Therefore there is no point creating space here. Moreover, that would lead to problems when a multi-row block with a position key such as `b` or `B`.


```

7236         }
7237     }
7238 }
7239 }
7240 \seq_gput_right:Nx \g_@@_blocks_seq
7241 {
7242     \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.

```

7243 {
7244     \exp_not:n { #3 } ,
7245     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7246     \bool_if:NT \g_@@_rotate_bool
7247     {
7248         \bool_if:NTF \g_@@_rotate_c_bool
7249         { v-center }
7250         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7251     }
7252 }
7253 {
7254     \box_use_drop:c
7255     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7256 }
7257 }
7258 \bool_set_false:N \g_@@_rotate_c_bool
7259 }
7260 }

```

```

7261 \cs_new:Npn \@@_adjust_hpos_rotate:
7262 {
7263     \bool_if:NT \g_@@_rotate_bool
7264     {
7265         \str_set:Nx \l_@@_hpos_block_str
7266         {
7267             \bool_if:NTF \g_@@_rotate_c_bool
7268             { c }
7269             {
7270                 \str_case:onF \l_@@_vpos_block_str
7271                 { b l B l t r T r }
7272                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7273             }
7274         }
7275     }
7276 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7277 \cs_new_protected:Npn \@@_rotate_box_of_block:
7278 {
7279     \box_grotate:cn
7280     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7281     { 90 }
7282     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7283     {
7284         \vbox_gset_top:cn
7285         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7286         {

```

```

7287         \skip_vertical:n { 0.8 ex }
7288         \box_use:c
7289         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7290     }
7291 }
7292 \bool_if:NT \g_@@_rotate_c_bool
7293 {
7294     \hbox_gset:cn
7295     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7296     {
7297         \c_math_toggle_token
7298         \vcenter
7299         {
7300             \box_use:c
7301             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7302         }
7303         \c_math_toggle_token
7304     }
7305 }
7306 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column. In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#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.

```

7307 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7308 {
7309     \seq_gput_right:Nx \g_@@_blocks_seq
7310     {
7311         \l_tmpa_tl
7312         { \exp_not:n { #3 } }
7313         {
7314             \bool_if:NTF \l_@@_tabular_bool
7315             {
7316                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7317         \@@_reset_arraystretch:
7318         \exp_not:n
7319         {
7320             \dim_zero:N \extrarowheight
7321             #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).

```

7322         \use:e
7323         {
7324             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7325             { @ { } \l_@@_hpos_block_str @ { } }
7326         }
7327         #5
7328         \end { tabular }
7329     }
7330     \group_end:
7331 }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7332         {
7333             \group_begin:
The following will be no-op when respect-arraystretch is in force.
7334             \@@_reset_arraystretch:
7335             \exp_not:n
7336             {
7337                 \dim_zero:N \extrarowheight
7338                 #4
7339                 \c_math_toggle_token
7340                 \use:e
7341                 {
7342                     \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7343                     { @ { } \l_@@_hpos_block_str @ { } }
7344                 }
7345                 #5
7346                 \end { array }
7347                 \c_math_toggle_token
7348             }
7349             \group_end:
7350         }
7351     }
7352 }
7353 }
```

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).

```

7354 \keys_define:nn { NiceMatrix / Block / SecondPass }
7355 {
7356     tikz .code:n =
7357         \IfPackageLoadedTF { tikz }
7358         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7359         { \@@_error:n { tikz~key~without~tikz } } ,
7360     tikz .value_required:n = true ,
7361     fill .code:n =
7362         \tl_set_rescan:Nnn
7363         \l_@@_fill_tl
7364         { \char_set_catcode_other:N ! }
7365         { #1 } ,
7366     fill .value_required:n = true ,
7367     opacity .tl_set:N = \l_@@_opacity_tl ,
7368     opacity .value_required:n = true ,
7369     draw .code:n =
7370         \tl_set_rescan:Nnn
7371         \l_@@_draw_tl
7372         { \char_set_catcode_other:N ! }
7373         { #1 } ,
7374     draw .default:n = default ,
7375     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7376     rounded-corners .default:n = 4 pt ,
7377     color .code:n =
7378         \@@_color:n { #1 }
7379         \tl_set_rescan:Nnn
7380         \l_@@_draw_tl
7381         { \char_set_catcode_other:N ! }
7382         { #1 } ,
7383     borders .clist_set:N = \l_@@_borders_clist ,
7384     borders .value_required:n = true ,
7385     hvlines .meta:n = { vlines , hlines } ,
7386     vlines .bool_set:N = \l_@@_vlines_block_bool,
```

```

7387     vl_lines .default:n = true ,
7388     hl_lines .bool_set:N = \l_@@_hl_lines_block_bool,
7389     hl_lines .default:n = true ,
7390     line_width .dim_set:N = \l_@@_line_width_dim ,
7391     line_width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7392     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7393     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7394     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7395     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7396         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7397     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7398         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7399     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7400         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7401     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7402     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7403     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7404     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7405     v-center .code:n = \str_set:Nn \l_@@_vpos_block_str { c } ,
7406     v-center .value_forbidden:n = true ,
7407     name .tl_set:N = \l_@@_block_name_str ,
7408     name .value_required:n = true ,
7409     name .initial:n = ,
7410     respect_arraystretch .code:n =
7411         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7412     respect_arraystretch .value_forbidden:n = true ,
7413     transparent .bool_set:N = \l_@@_transparent_bool ,
7414     transparent .default:n = true ,
7415     transparent .initial:n = false ,
7416     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7417 }

```

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.

```

7418 \cs_new_protected:Npn \@@_draw_blocks:
7419 {
7420     \cs_set_eq:NN \ialign \@@_old_ialign:
7421     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7422 }
7423 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7424 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7425     \int_zero_new:N \l_@@_last_row_int
7426     \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.

```

7427     \int_compare:nNnTF { #3 } > { 99 }
7428     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7429     { \int_set:Nn \l_@@_last_row_int { #3 } }
7430     \int_compare:nNnTF { #4 } > { 99 }
7431     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7432     { \int_set:Nn \l_@@_last_col_int { #4 } }
7433     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int

```

```

7434 {
7435   \bool_lazy_and:nnTF
7436   \l_@@_preamble_bool
7437   { \int_compare:nNnF \l_@@_last_col_int > \g_@@_static_num_of_col_int }
7438   {
7439     \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7440     \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7441     \@@_msg_redirect_name:nn { columns-not-used } { none }
7442   }
7443   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7444 }
7445 {
7446   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7447   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7448   { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7449 }
7450 }

```

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

```

7451 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7452 {

```

The group is for the keys.

```

7453   \group_begin:
7454   \int_compare:nNnT { #1 } = { #3 }
7455   { \str_set:Nn \l_@@_vpos_block_str { t } }
7456   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7457   \bool_if:NT \l_@@_vlines_block_bool
7458   {
7459     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7460     {
7461       \@@_vlines_block:nnn
7462       { \exp_not:n { #5 } }
7463       { #1 - #2 }
7464       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7465     }
7466   }
7467   \bool_if:NT \l_@@_hlines_block_bool
7468   {
7469     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7470     {
7471       \@@_hlines_block:nnn
7472       { \exp_not:n { #5 } }
7473       { #1 - #2 }
7474       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7475     }
7476   }
7477   \bool_if:NF \l_@@_transparent_bool
7478   {
7479     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7480     {

```

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).

```

7481       \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7482       { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7483     }
7484   }

7485   \tl_if_empty:NF \l_@@_draw_tl
7486   {

```

```

7487 \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7488 { \@@_error:n { hlines-with-color } }
7489 }

7490 \tl_if_empty:NF \l_@@_draw_tl
7491 {
7492   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7493   {
7494     \@@_stroke_block:nnn
7495     { \exp_not:n { #5 } } } % #5 are the options
7496     { #1 - #2 }
7497     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7498   }
7499   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7500   { { #1 } { #2 } { #3 } { #4 } }
7501 }

7502 \clist_if_empty:NF \l_@@_borders_clist
7503 {
7504   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7505   {
7506     \@@_stroke_borders_block:nnn
7507     { \exp_not:n { #5 } }
7508     { #1 - #2 }
7509     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7510   }
7511 }

7512 \tl_if_empty:NF \l_@@_fill_tl
7513 {
7514   \tl_if_empty:NF \l_@@_opacity_tl
7515   {
7516     \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7517       {
7518         \tl_set:Nx \l_@@_fill_tl
7519         {
7520           [ opacity = \l_@@_opacity_tl ,
7521             \tl_tail:o \l_@@_fill_tl
7522         ]
7523       }
7524       {
7525         \tl_set:Nx \l_@@_fill_tl
7526         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7527       }
7528     }
7529     \tl_gput_right:Nx \g_@@_pre_code_before_tl
7530     {
7531       \exp_not:N \roundedrectanglecolor
7532       \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7533         { \l_@@_fill_tl }
7534         { { \l_@@_fill_tl } }
7535         { #1 - #2 }
7536         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7537         { \dim_use:N \l_@@_rounded_corners_dim }
7538       }
7539     }
7540   }

7540 \seq_if_empty:NF \l_@@_tikz_seq
7541 {
7542   \tl_gput_right:Nx \g_nicematrix_code_before_tl
7543   {
7544     \@@_block_tikz:nnnnn
7545     { #1 }
7546     { #2 }
7547     { \int_use:N \l_@@_last_row_int }

```

```

7548         { \int_use:N \l_@@_last_col_int }
7549         { \seq_use:Nn \l_@@_tikz_seq { , } }
7550     }
7551 }

7552 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7553 {
7554     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7555     {
7556         \@@_actually_diagbox:nnnnnn
7557         { #1 }
7558         { #2 }
7559         { \int_use:N \l_@@_last_row_int }
7560         { \int_use:N \l_@@_last_col_int }
7561         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7562     }
7563 }

7564 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7565 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one      & \\
                        &      & two      & \\
three                  & four & five      & \\
six                    & seven & eight      & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

7566 \pgfpicture
7567 \pgfrememberpicturepositiononpagetrue
7568 \pgf@relevantforpicturesizefalse
7569 \@@_qpoint:n { row - #1 }
7570 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7571 \@@_qpoint:n { col - #2 }
7572 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7573 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7574 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7575 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7576 \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.

```

7577 \@@_pgf_rect_node:nnnnn
7578 { \@@_env: - #1 - #2 - block }
7579 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7580 \str_if_empty:NF \l_@@_block_name_str
7581 {
7582     \pgfnodealias

```

```

7583     { \@@_env: - \l_@@_block_name_str }
7584     { \@@_env: - #1 - #2 - block }
7585   \str_if_empty:NF \l_@@_name_str
7586   {
7587     \pgfnodealias
7588     { \l_@@_name_str - \l_@@_block_name_str }
7589     { \@@_env: - #1 - #2 - block }
7590   }
7591 }

```

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.

```

7592   \bool_if:NF \l_@@_hpos_of_block_cap_bool
7593   {
7594     \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.

```

7595     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7596     {

```

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.

```

7597       \cs_if_exist:cT
7598       { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7599       {
7600         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7601         {
7602           \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7603           \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7604         }
7605       }
7606     }

```

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.

```

7607     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7608     {
7609       \@@_qpoint:n { col - #2 }
7610       \dim_set_eq:NN \l_tmpb_dim \pgf@x
7611     }
7612   \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7613   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7614   {
7615     \cs_if_exist:cT
7616     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7617     {
7618       \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7619       {
7620         \pgfpointanchor
7621         { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7622         { east }
7623         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7624       }
7625     }
7626   }
7627   \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7628   {
7629     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7630     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7631   }
7632   \@@_pgf_rect_node:nnnnn

```



```

7633         { \@@_env: - #1 - #2 - block - short }
7634         \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7635     }

```

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.

```

7636     \bool_if:NT \l_@@_medium_nodes_bool
7637     {
7638         \@@_pgf_rect_node:nnn
7639         { \@@_env: - #1 - #2 - block - medium }
7640         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7641         {
7642             \pgfpointanchor
7643             { \@@_env:
7644                 - \int_use:N \l_@@_last_row_int
7645                 - \int_use:N \l_@@_last_col_int - medium
7646             }
7647             { south-east }
7648         }
7649     }

```

Now, we will put the label of the block.

```

7650     \bool_lazy_any:nTF
7651     {
7652         { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7653         { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7654         { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7655     }
7656     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7657         \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`.

```

7658         \bool_if:nT \g_@@_last_col_found_bool
7659         {
7660             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7661             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7662         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7663         \tl_set:Nx \l_tmpa_tl
7664         {
7665             \str_case:on \l_@@_vpos_block_str
7666             {
7667                 c {
7668                     \str_case:on \l_@@_hpos_block_str
7669                     {
7670                         c { center }
7671                         l { west }
7672                         r { east }
7673                     }
7674                 }
7675             }
7676             T {
7677                 \str_case:on \l_@@_hpos_block_str
7678                 {
7679                     c { north }
7680                     l { north-west }
7681                     r { north-east }
7682                 }
7683             }
7684         }

```

```

7685         B {
7686             \str_case:on \l_@@_hpos_block_str
7687             {
7688                 c { south}
7689                 l { south~west }
7690                 r { south~east }
7691             }
7692
7693         }
7694     }
7695 }
7696 \pgftransformshift
7697 {
7698     \pgfpointanchor
7699     {
7700         \@@_env: - #1 - #2 - block
7701         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7702     }
7703     { \l_tmpa_tl }
7704 }
7705 \pgfset
7706 {
7707     inner~xsep = \c_zero_dim ,
7708     inner~ysep = \c_zero_dim
7709 }
7710 \pgfnode
7711 { rectangle }
7712 { \l_tmpa_tl }
7713 { \box_use_drop:N \l_@@_cell_box } { } { }
7714 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

7715 {
7716     \pgfextracty \l_tmpa_dim
7717     {
7718         \@@_qpoint:n
7719         {
7720             row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7721             - base
7722         }
7723     }
7724     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth } % added 2023-02-21

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

7725 \pgfpointanchor
7726 {
7727     \@@_env: - #1 - #2 - block
7728     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7729 }
7730 {
7731     \str_case:on \l_@@_hpos_block_str
7732     {
7733         c { center }
7734         l { west }
7735         r { east }
7736     }
7737 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7738 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7739 \pgfset { inner~sep = \c_zero_dim }
7740 \pgfnode
7741 { rectangle }
7742 {

```

```

7743         \str_case:on \l_@@_hpos_block_str
7744         {
7745             c { base }
7746             l { base~west }
7747             r { base~east }
7748         }
7749     }
7750     { \box_use_drop:N \l_@@_cell_box } { } { }
7751 }
7752 \endpgfpicture
7753 \group_end:
7754 }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

7755 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7756 {
7757     \group_begin:
7758     \tl_clear:N \l_@@_draw_tl
7759     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7760     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7761     \pgfpicture
7762     \pgfrememberpicturepositiononpagetrue
7763     \pgf@relevantforpicturesizefalse
7764     \tl_if_empty:NF \l_@@_draw_tl
7765     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7766         \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7767         { \CT@arc@ }
7768         { \@@_color:o \l_@@_draw_tl }
7769     }
7770     \pgfsetcornersarced
7771     {
7772         \pgfpoint
7773         { \l_@@_rounded_corners_dim }
7774         { \l_@@_rounded_corners_dim }
7775     }
7776     \@@_cut_on_hyphen:w #2 \q_stop
7777     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7778     {
7779         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7780         {
7781             \@@_qpoint:n { row - \l_tmpa_tl }
7782             \dim_set_eq:NN \l_tmpb_dim \pgf@y
7783             \@@_qpoint:n { col - \l_tmpb_tl }
7784             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7785             \@@_cut_on_hyphen:w #3 \q_stop
7786             \int_compare:nNnT \l_tmpa_tl > \c@iRow
7787             { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7788             \int_compare:nNnT \l_tmpb_tl > \c@jCol
7789             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7790             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7791             \dim_set_eq:NN \l_tmpa_dim \pgf@y
7792             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7793             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7794             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7795             \pgfpathrectanglecorners
7796             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7797             { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7798             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim

```

```

7799         { \pgfusepathqstroke }
7800         { \pgfusepath { stroke } }
7801     }
7802 }
7803 \endpgfpicture
7804 \group_end:
7805 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7806 \keys_define:nn { NiceMatrix / BlockStroke }
7807 {
7808     color .tl_set:N = \l_@@_draw_tl ,
7809     draw .code:n =
7810         \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7811     draw .default:n = default ,
7812     line-width .dim_set:N = \l_@@_line_width_dim ,
7813     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7814     rounded-corners .default:n = 4 pt
7815 }

```

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).

```

7816 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7817 {
7818     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7819     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7820     \@@_cut_on_hyphen:w #2 \q_stop
7821     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7822     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7823     \@@_cut_on_hyphen:w #3 \q_stop
7824     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7825     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7826     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7827     {
7828         \use:e
7829         {
7830             \@@_vline:n
7831             {
7832                 position = ##1 ,
7833                 start = \l_@@_tmpc_tl ,
7834                 end = \int_eval:n { \l_tmpa_tl - 1 } ,
7835                 total-width = \dim_use:N \l_@@_line_width_dim
7836             }
7837         }
7838     }
7839 }
7840 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7841 {
7842     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7843     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7844     \@@_cut_on_hyphen:w #2 \q_stop
7845     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7846     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7847     \@@_cut_on_hyphen:w #3 \q_stop
7848     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7849     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7850     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7851     {
7852         \use:e
7853         {
7854             \@@_hline:n
7855             {

```

```

7856         position = ##1 ,
7857         start = \l_@@_tmpd_tl ,
7858         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7859         total-width = \dim_use:N \l_@@_line_width_dim
7860     }
7861 }
7862 }
7863 }

```

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).

```

7864 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7865 {
7866     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7867     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7868     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7869     { \@@_error:n { borders~forbidden } }
7870     {
7871         \tl_clear_new:N \l_@@_borders_tikz_tl
7872         \keys_set:nV
7873             { NiceMatrix / OnlyForTikzInBorders }
7874         \l_@@_borders_clist
7875         \@@_cut_on_hyphen:w #2 \q_stop
7876         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7877         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7878         \@@_cut_on_hyphen:w #3 \q_stop
7879         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7880         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7881         \@@_stroke_borders_block_i:
7882     }
7883 }
7884 \hook_gput_code:nnn { begindocument } { . }
7885 {
7886     \cs_new_protected:Npx \@@_stroke_borders_block_i:
7887     {
7888         \c_@@_pgfortikzpicture_tl
7889         \@@_stroke_borders_block_ii:
7890         \c_@@_endpgfortikzpicture_tl
7891     }
7892 }
7893 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7894 {
7895     \pgfrememberpicturepositiononpagetrue
7896     \pgf@relevantforpicturesizefalse
7897     \CT@arc@
7898     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7899     \clist_if_in:NnT \l_@@_borders_clist { right }
7900     { \@@_stroke_vertical:n \l_tmpb_tl }
7901     \clist_if_in:NnT \l_@@_borders_clist { left }
7902     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7903     \clist_if_in:NnT \l_@@_borders_clist { bottom }
7904     { \@@_stroke_horizontal:n \l_tmpa_tl }
7905     \clist_if_in:NnT \l_@@_borders_clist { top }
7906     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7907 }
7908 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7909 {
7910     tikz .code:n =
7911         \cs_if_exist:NTF \tikzpicture
7912         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7913         { \@@_error:n { tikz~in~borders~without~tikz } } ,

```

```

7914     tikz .value_required:n = true ,
7915     top .code:n = ,
7916     bottom .code:n = ,
7917     left .code:n = ,
7918     right .code:n = ,
7919     unknown .code:n = \@@_error:n { bad-border }
7920 }

```

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).

```

7921 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7922 {
7923   \@@_qpoint:n \l_@@_tmpc_tl
7924   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7925   \@@_qpoint:n \l_tmpa_tl
7926   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7927   \@@_qpoint:n { #1 }
7928   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7929   {
7930     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7931     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7932     \pgfusepathqstroke
7933   }
7934   {
7935     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7936     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7937   }
7938 }

```

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).

```

7939 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7940 {
7941   \@@_qpoint:n \l_@@_tmpd_tl
7942   \clist_if_in:NnTF \l_@@_borders_clist { left }
7943   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7944   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7945   \@@_qpoint:n \l_tmpb_tl
7946   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7947   \@@_qpoint:n { #1 }
7948   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7949   {
7950     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7951     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7952     \pgfusepathqstroke
7953   }
7954   {
7955     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7956     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7957   }
7958 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

7959 \keys_define:nn { NiceMatrix / BlockBorders }
7960 {
7961   borders .clist_set:N = \l_@@_borders_clist ,
7962   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7963   rounded-corners .default:n = 4 pt ,
7964   line-width .dim_set:N = \l_@@_line_width_dim
7965 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments `#1` and `#2` are the coordinates of the first cell and `#3` and `#4` the coordinates of the last cell of the block. `#5` is a comma-separated list of the Tikz keys used with the path. However, among those keys, you have added in `nicematrix` a special key `offset` (an offset for the rectangle of the block). That's why we have to extract that key first.

```

7966 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7967 {
7968   \begin { tikzpicture }
7969   \@@_clip_with_rounded_corners:
7970   \clist_map_inline:nn { #5 }
7971   {
7972     \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7973     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7974     (
7975       [
7976         xshift = \dim_use:N \l_@@_offset_dim ,
7977         yshift = - \dim_use:N \l_@@_offset_dim
7978       ]
7979       #1 -| #2
7980     )
7981     rectangle
7982     (
7983       [
7984         xshift = - \dim_use:N \l_@@_offset_dim ,
7985         yshift = \dim_use:N \l_@@_offset_dim
7986       ]
7987       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7988     ) ;
7989   }
7990   \end { tikzpicture }
7991 }
7992 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7993 \keys_define:nn { NiceMatrix / SpecialOffset }
7994 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7995 \cs_set_protected:Npn \@@_renew_matrix:
7996 {
7997   \RenewDocumentEnvironment { pmatrix } { } {
7998     { \pNiceMatrix }
7999     { \endpNiceMatrix }
8000   }
8001   \RenewDocumentEnvironment { vmatrix } { } {
8002     { \vNiceMatrix }
8003     { \endvNiceMatrix }
8004   }
8005   \RenewDocumentEnvironment { Vmatrix } { } {
8006     { \VNiceMatrix }
8007     { \endVNiceMatrix }
8008   }
8009   \RenewDocumentEnvironment { bmatrix } { } {
8010     { \bNiceMatrix }
8011     { \endbNiceMatrix }
8012   }
8013   \RenewDocumentEnvironment { Bmatrix } { } {
8014     { \BNiceMatrix }
8015     { \endBNiceMatrix }
8016   }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8013 \keys_define:nn { NiceMatrix / Auto }
8014 {
8015   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8016   columns-type .value_required:n = true ,
8017   l .meta:n = { columns-type = l } ,
8018   r .meta:n = { columns-type = r } ,
8019   c .meta:n = { columns-type = c } ,
8020   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8021   delimiters / color .value_required:n = true ,
8022   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8023   delimiters / max-width .default:n = true ,
8024   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
8025   delimiters .value_required:n = true ,
8026   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8027   rounded-corners .default:n = 4 pt
8028 }

8029 \NewDocumentCommand \AutoNiceMatrixWithDelims
8030 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8031 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8032 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8033 {

```

The group is for the protection of the keys.

```

8034   \group_begin:
8035   \keys_set:known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
8036   \use:e
8037   {
8038     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8039     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8040     [ \exp_not:o \l_tmpa_tl ]
8041   }
8042   \int_if_zero:nT \l_@@_first_row_int
8043   {
8044     \int_if_zero:nT \l_@@_first_col_int { & }
8045     \prg_replicate:nn { #4 - 1 } { & }
8046     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8047   }
8048   \prg_replicate:nn { #3 }
8049   {
8050     \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`).

```

8051     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8052     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8053   }
8054   \int_compare:nNnT \l_@@_last_row_int > { -2 }
8055   {
8056     \int_if_zero:nT \l_@@_first_col_int { & }
8057     \prg_replicate:nn { #4 - 1 } { & }
8058     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8059   }
8060   \end { NiceArrayWithDelims }
8061   \group_end:
8062 }

8063 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8064 {

```



```

8065 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8066 {
8067   \bool_gset_true:N \g_@@_delims_bool
8068   \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8069   \AutoNiceMatrixWithDelims { #2 } { #3 }
8070 }
8071 }

8072 \@@_define_com:nnn p ( )
8073 \@@_define_com:nnn b [ ]
8074 \@@_define_com:nnn v | |
8075 \@@_define_com:nnn V \! \!
8076 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8077 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8078 {
8079   \group_begin:
8080   \bool_gset_false:N \g_@@_delims_bool
8081   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8082   \group_end:
8083 }

```

30 The redefinition of the command `\dotfill`

```

8084 \cs_set_eq:NN \@@_old_dotfill \dotfill
8085 \cs_new_protected:Npn \@@_dotfill:
8086 {

```

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}`).

```

8087   \@@_old_dotfill
8088   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8089 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8090 \cs_new_protected:Npn \@@_dotfill_i:
8091 { \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.

```

8092 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8093 {
8094   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8095   {
8096     \@@_actually_diagbox:nnnnnn
8097     { \int_use:N \c@iRow }
8098     { \int_use:N \c@jCol }
8099     { \int_use:N \c@iRow }
8100     { \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.

```
8101         { \g_@@_row_style_tl \exp_not:n { #1 } }
8102         { \g_@@_row_style_tl \exp_not:n { #2 } }
8103     }
```

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`.

```
8104     \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8105     {
8106         { \int_use:N \c@iRow }
8107         { \int_use:N \c@jCol }
8108         { \int_use:N \c@iRow }
8109         { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8110         { }
8111     }
8112 }
```

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.

```
8113 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8114 {
8115     \pgfpicture
8116     \pgf@relevantforpicturesizefalse
8117     \pgfrememberpicturepositiononpagetrue
8118     \@@_qpoint:n { row - #1 }
8119     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8120     \@@_qpoint:n { col - #2 }
8121     \dim_set_eq:NN \l_tmpb_dim \pgf@x
8122     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8123     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8124     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8125     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8126     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8127     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8128 }
```

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.

```
8129     \CT@arc@
8130     \pgfsetroundcap
8131     \pgfusepathqstroke
8132 }
8133 \pgfset { inner~sep = 1 pt }
8134 \pgfscope
8135 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8136 \pgfnode { rectangle } { south~west }
8137 {
8138     \begin { minipage } { 20 cm }
8139     \@@_math_toggle: #5 \@@_math_toggle:
8140     \end { minipage }
8141 }
8142 { }
8143 { }
8144 \endpgfscope
8145 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
```

```

8146 \pgfnode { rectangle } { north-east }
8147 {
8148   \begin { minipage } { 20 cm }
8149   \raggedleft
8150   \@@_math_toggle: #6 \@@_math_toggle:
8151   \end { minipage }
8152 }
8153 { }
8154 { }
8155 \endpgfpicture
8156 }

```

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. 83.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8157 \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 `\`.

```

8158 \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`.

```

8159 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8160 {
8161   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8162   \@@_CodeAfter_iv:n
8163 }

```

We catch the argument of the command `\end` (in `#1`).

```

8164 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8165 {

```

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.

```

8166   \str_if_eq:eeTF \@currenvir { #1 }
8167   { \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`.

```

8168   {
8169     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8170     \@@_CodeAfter_ii:n
8171   }
8172 }

```

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.

```

8173 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8174 {
8175   \pgfpicture
8176   \pgfrememberpicturepositiononpagetrue
8177   \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.

```

8178   \@@_qpoint:n { row - 1 }
8179   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8180   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8181   \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).

```

8182   \bool_if:nTF { #3 }
8183   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8184   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8185   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8186   {
8187     \cs_if_exist:cT
8188     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8189     {
8190       \pgfpointanchor
8191       { \@@_env: - ##1 - #2 }
8192       { \bool_if:nTF { #3 } { west } { east } }
8193       \dim_set:Nn \l_tmpa_dim
8194       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8195     }
8196   }

```

Now we can put the delimiter with a node of PGF.

```

8197   \pgfset { inner~sep = \c_zero_dim }
8198   \dim_zero:N \nulldelimiterspace
8199   \pgftransformshift
8200   {
8201     \pgfpoint
8202     { \l_tmpa_dim }
8203     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8204   }
8205   \pgfnode
8206   { rectangle }
8207   { \bool_if:nTF { #3 } { east } { west } }
8208   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8209   \nullfont
8210   \c_math_toggle_token
8211   \@@_color:o \l_@@_delimiters_color_tl
8212   \bool_if:nTF { #3 } { \left #1 } { \left . }

```

```

8213     \vcenter
8214     {
8215         \nullfont
8216         \hrule \@height
8217             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8218             \@depth \c_zero_dim
8219             \@width \c_zero_dim
8220     }
8221     \bool_if:nTF { #3 } { \right . } { \right #1 }
8222     \c_math_toggle_token
8223 }
8224 { }
8225 { }
8226 \endpgfpicture
8227 }

```

34 The command \SubMatrix

```

8228 \keys_define:nn { NiceMatrix / sub-matrix }
8229 {
8230     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8231     extra-height .value_required:n = true ,
8232     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8233     left-xshift .value_required:n = true ,
8234     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8235     right-xshift .value_required:n = true ,
8236     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8237     xshift .value_required:n = true ,
8238     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8239     delimiters / color .value_required:n = true ,
8240     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8241     slim .default:n = true ,
8242     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8243     hlines .default:n = all ,
8244     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8245     vlines .default:n = all ,
8246     hvlines .meta:n = { hlines, vlines } ,
8247     hvlines .value_forbidden:n = true
8248 }
8249 \keys_define:nn { NiceMatrix }
8250 {
8251     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8252     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8253     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8254     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8255 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8256 \keys_define:nn { NiceMatrix / SubMatrix }
8257 {
8258     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8259     delimiters / color .value_required:n = true ,
8260     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8261     hlines .default:n = all ,
8262     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8263     vlines .default:n = all ,
8264     hvlines .meta:n = { hlines, vlines } ,
8265     hvlines .value_forbidden:n = true ,
8266     name .code:n =

```

```

8267 \tl_if_empty:nTF { #1 }
8268 { \@@_error:n { Invalid-name } }
8269 {
8270   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8271   {
8272     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8273     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8274     {
8275       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8276       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8277     }
8278   }
8279   { \@@_error:n { Invalid-name } }
8280 } ,
8281 name .value_required:n = true ,
8282 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8283 rules .value_required:n = true ,
8284 code .tl_set:N = \l_@@_code_tl ,
8285 code .value_required:n = true ,
8286 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8287 }

8288 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8289 {
8290   \peek_remove_spaces:n
8291   {
8292     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8293     {
8294       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8295       [
8296         delimiters / color = \l_@@_delimiters_color_tl ,
8297         hlines = \l_@@_submatrix_hlines_clist ,
8298         vlines = \l_@@_submatrix_vlines_clist ,
8299         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8300         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8301         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8302         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8303         #5
8304       ]
8305     }
8306     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8307   }
8308 }

8309 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8310 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8311 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8312 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8313 {
8314   \seq_gput_right:Nx \g_@@_submatrix_seq
8315   {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8316 { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8317 { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8318 { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8319 { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8320 }
8321 }

```

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`.

```

8322 \hook_gput_code:nnn { begindocument } { . }
8323 {
8324   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8325   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8326   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8327     {
8328       \peek_remove_spaces:n
8329       {
8330         \@@_sub_matrix:nnnnnnn
8331         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8332       }
8333     }
8334 }

```

The following macro will compute `\l_@@_first_i_tl`, `\l_@@_first_j_tl`, `\l_@@_last_i_tl` and `\l_@@_last_j_tl` from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

8335 \NewDocumentCommand \@@_compute_i_j:nn
8336 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8337 { \@@_compute_i_j:nnnn #1 #2 }
8338 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8339 {
8340   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8341   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8342   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8343   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8344   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8345     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8346   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8347     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8348   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8349     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8350   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8351     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8352 }
8353 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8354 {
8355   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8356 \@@_compute_i_j:nn { #2 } { #3 }
8357 \int_compare:NnNT \l_@@_first_i_tl = \l_@@_last_i_tl
8358   { \cs_set_nopar:Npn \arraystretch { 1 } }
8359 \bool_lazy_or:nnTF
8360   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8361   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8362   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8363   {
8364     \str_clear_new:N \l_@@_submatrix_name_str
8365     \keys_set:nn { NiceMatrix / SubMatrix } { #5 }

```

```

8366 \pgfpicture
8367 \pgfrememberpicturepositiononpagetrue
8368 \pgf@relevantforpicturesizefalse
8369 \pgfset { inner~sep = \c_zero_dim }
8370 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8371 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currification.

8372 \bool_if:NTF \l_@@_submatrix_slim_bool
8373 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8374 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8375 {
8376   \cs_if_exist:cT
8377   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8378   {
8379     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8380     \dim_set:Nn \l_@@_x_initial_dim
8381     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8382   }
8383   \cs_if_exist:cT
8384   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8385   {
8386     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8387     \dim_set:Nn \l_@@_x_final_dim
8388     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8389   }
8390 }
8391 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8392 { \@@_error:nn { Impossible~delimiter } { left } }
8393 {
8394   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8395   { \@@_error:nn { Impossible~delimiter } { right } }
8396   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8397 }
8398 \endpgfpicture
8399 }
8400 \group_end:
8401 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8402 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8403 {
8404   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8405   \dim_set:Nn \l_@@_y_initial_dim
8406   {
8407     \fp_to_dim:n
8408     {
8409       \pgf@y
8410       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8411     }
8412     } % modified 6.13c
8413   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8414   \dim_set:Nn \l_@@_y_final_dim
8415   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8416   } % modified 6.13c
8417   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8418   {
8419     \cs_if_exist:cT
8420     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8421     {
8422       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8423       \dim_set:Nn \l_@@_y_initial_dim
8424       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }

```



```

8425     }
8426     \cs_if_exist:cT
8427     { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8428     {
8429         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8430         \dim_set:Nn \l_@@_y_final_dim
8431         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8432     }
8433 }
8434 \dim_set:Nn \l_tmpa_dim
8435 {
8436     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8437     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8438 }
8439 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8440     \group_begin:
8441     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8442     \@@_set_CT@arc@:o \l_@@_rules_color_tl
8443     \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`.

```

8444     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8445     {
8446         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8447         {
8448             \int_compare:nNnT
8449             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8450             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8451             \@@_qpoint:n { col - ##1 }
8452             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8453             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8454             \pgfusepathqstroke
8455         }
8456     }
8457 }

```

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.

```

8458     \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8459     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8460     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8461     {
8462         \bool_lazy_and:nnTF
8463         { \int_compare_p:nNn { ##1 } > \c_zero_int }
8464         {
8465             \int_compare_p:nNn
8466             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8467         {
8468             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8469             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8470             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8471             \pgfusepathqstroke
8472         }
8473         { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8474     }

```

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.

```

8475 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8476 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8477 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8478 {
8479   \bool_lazy_and:nnTF
8480   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8481   {
8482     \int_compare_p:nNn
8483     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8484   {
8485     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8486   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8487   \dim_set:Nn \l_tmpa_dim
8488   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8489   \str_case:nn { #1 }
8490   {
8491     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8492     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8493     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8494     }
8495     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

8496   \dim_set:Nn \l_tmpb_dim
8497   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8498   \str_case:nn { #2 }
8499   {
8500     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8501     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8502     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8503   }
8504   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8505   \pgfusepathqstroke
8506   \group_end:
8507 }
8508 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8509 }

```

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).

```

8510 \str_if_empty:NF \l_@@_submatrix_name_str
8511 {
8512   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8513   \l_@@_x_initial_dim \l_@@_y_initial_dim
8514   \l_@@_x_final_dim \l_@@_y_final_dim
8515 }
8516 \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`.

```

8517 \begin { pgfscope }
8518 \pgftransformshift
8519 {
8520   \pgfpoint
8521   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8522   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8523 }

```

```

8524 \str_if_empty:NTF \l_@@_submatrix_name_str
8525 { \@@_node_left:nn #1 { } }
8526 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8527 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8528 \pgftransformshift
8529 {
8530   \pgfpoint
8531   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8532   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8533 }
8534 \str_if_empty:NTF \l_@@_submatrix_name_str
8535 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8536 {
8537   \@@_node_right:nnnn #2
8538   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8539 }
8540 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8541 \flag_clear_new:n { nicematrix }
8542 \l_@@_code_tl
8543 }

```

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`.

```

8544 \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.

```

8545 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8546 {
8547   \use:e
8548   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8549 }

```

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 -.

```

8550 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8551 { #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.

```

8552 \tl_const:Nn \c_@@_integers_alist_tl
8553 {
8554   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8555   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8556   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8557   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8558 }
8559 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8560 {

```

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`.

```

8561 \tl_if_empty:nTF { #2 }
8562 {
8563   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8564   {
8565     \flag_raise:n { nicematrix }
8566     \int_if_even:nTF { \flag_height:n { nicematrix } }
8567     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8568     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8569   }
8570   { #1 }
8571 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8572 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8573 }

```

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`).

```

8574 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8575 {
8576   \str_case:nnF { #1 }
8577   {
8578     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8579     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8580   }

```

Now the case of a node of the form $i-j$.

```

8581 {
8582   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8583   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8584 }
8585 }

```

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`).

```

8586 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8587 {
8588   \pgfnode
8589   { rectangle }
8590   { east }
8591   {
8592     \nullfont
8593     \c_math_toggle_token
8594     \@@_color:o \l_@@_delimiters_color_tl
8595     \left #1
8596     \vcenter
8597     {
8598       \nullfont
8599       \hrule \@height \l_tmpa_dim
8600       \@depth \c_zero_dim
8601       \@width \c_zero_dim
8602     }
8603     \right .
8604     \c_math_toggle_token
8605   }
8606   { #2 }

```

```

8607 { }
8608 }

```

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.

```

8609 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8610 {
8611   \pgfnode
8612     { rectangle }
8613     { west }
8614     {
8615       \nullfont
8616       \c_math_toggle_token
8617       \@@_color:o \l_@@_delimiters_color_tl
8618       \left .
8619       \vcenter
8620         {
8621           \nullfont
8622           \hrule \@height \l_tmpa_dim
8623             \@depth \c_zero_dim
8624             \@width \c_zero_dim
8625         }
8626       \right #1
8627       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8628       ^ { \smash { #4 } }
8629       \c_math_toggle_token
8630     }
8631     { #2 }
8632     { }
8633 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8634 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8635 {
8636   \peek_remove_spaces:n
8637   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8638 }
8639 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8640 {
8641   \peek_remove_spaces:n
8642   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8643 }
8644 \keys_define:nn { NiceMatrix / Brace }
8645 {
8646   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8647   left-shorten .default:n = true ,
8648   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8649   shorten .meta:n = { left-shorten , right-shorten } ,
8650   right-shorten .default:n = true ,
8651   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8652   yshift .value_required:n = true ,
8653   yshift .initial:n = \c_zero_dim ,
8654   color .tl_set:N = \l_tmpa_tl ,
8655   color .value_required:n = true ,

```

```

8656     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8657 }

```

#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`.

```

8658 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8659 {
8660   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8661   \@@_compute_i_j:nn { #1 } { #2 }
8662   \bool_lazy_or:nnTF
8663     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8664     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8665   {
8666     \str_if_eq:nnTF { #5 } { under }
8667     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8668     { \@@_error:nn { Construct~too~large } { \OverBrace } }
8669   }
8670   {
8671     \tl_clear:N \l_tmpa_tl
8672     \keys_set:nn { NiceMatrix / Brace } { #4 }
8673     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8674     \pgfpicture
8675     \pgfrememberpicturepositiononpagetrue
8676     \pgf@relevantforpicturesizefalse
8677     \bool_if:NT \l_@@_brace_left_shorten_bool
8678     {
8679       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8680       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8681       {
8682         \cs_if_exist:cT
8683           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8684         {
8685           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8686           \dim_set:Nn \l_@@_x_initial_dim
8687             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8688         }
8689       }
8690     }
8691     \bool_lazy_or:nnT
8692       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8693       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8694     {
8695       \@@_qpoint:n { col - \l_@@_first_j_tl }
8696       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8697     }
8698     \bool_if:NT \l_@@_brace_right_shorten_bool
8699     {
8700       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8701       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8702       {
8703         \cs_if_exist:cT
8704           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8705         {
8706           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8707           \dim_set:Nn \l_@@_x_final_dim
8708             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8709         }
8710       }
8711     }
8712     \bool_lazy_or:nnT
8713       { \bool_not_p:n \l_@@_brace_right_shorten_bool }

```

```

8714         { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8715         {
8716             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8717             \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8718         }
8719         \pgfset { inner~sep = \c_zero_dim }
8720         \str_if_eq:nnTF { #5 } { under }
8721             { \@@_underbrace_i:n { #3 } }
8722             { \@@_overbrace_i:n { #3 } }
8723         \endpgfpicture
8724     }
8725 \group_end:
8726 }

```

The argument is the text to put above the brace.

```

8727 \cs_new_protected:Npn \@@_overbrace_i:n #1
8728 {
8729     \@@_qpoint:n { row - \l_@@_first_i_tl }
8730     \pgftransformshift
8731     {
8732         \pgfpoint
8733         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8734         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8735     }
8736     \pgfnode
8737     { rectangle }
8738     { south }
8739     {
8740         \vtop
8741         {
8742             \group_begin:
8743             \everycr { }
8744             \halign
8745             {
8746                 \hfil ## \hfil \crcr
8747                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8748                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8749                 \c_math_toggle_token
8750                 \overbrace
8751                 {
8752                     \hbox_to_wd:nn
8753                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8754                     { }
8755                 }
8756                 \c_math_toggle_token
8757                 \cr
8758             }
8759             \group_end:
8760         }
8761     }
8762     { }
8763     { }
8764 }

```

The argument is the text to put under the brace.

```

8765 \cs_new_protected:Npn \@@_underbrace_i:n #1
8766 {
8767     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8768     \pgftransformshift
8769     {
8770         \pgfpoint
8771         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8772         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }

```

```

8773     }
8774 \pgfnode
8775 { rectangle }
8776 { north }
8777 {
8778   \group_begin:
8779   \everycr { }
8780   \vbox
8781   {
8782     \halign
8783     {
8784       \hfil ## \hfil \crcr
8785       \c_math_toggle_token
8786       \underbrace
8787       {
8788         \hbox_to_wd:nn
8789         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8790         { }
8791       }
8792       \c_math_toggle_token
8793       \cr
8794       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8795       \@@_math_toggle: #1 \@@_math_toggle: \cr
8796     }
8797   }
8798   \group_end:
8799 }
8800 { }
8801 { }
8802 }

```

36 The command TikzEveryCell

```

8803 \bool_new:N \l_@@_not_empty_bool
8804 \bool_new:N \l_@@_empty_bool
8805
8806 \keys_define:nn { NiceMatrix / TikzEveryCell }
8807 {
8808   not-empty .code:n =
8809     \bool_lazy_or:nnTF
8810     \l_@@_in_code_after_bool
8811     \g_@@_recreate_cell_nodes_bool
8812     { \bool_set_true:N \l_@@_not_empty_bool }
8813     { \@@_error:n { detection~of~empty~cells } } ,
8814   not-empty .value_forbidden:n = true ,
8815   empty .code:n =
8816     \bool_lazy_or:nnTF
8817     \l_@@_in_code_after_bool
8818     \g_@@_recreate_cell_nodes_bool
8819     { \bool_set_true:N \l_@@_empty_bool }
8820     { \@@_error:n { detection~of~empty~cells } } ,
8821   empty .value_forbidden:n = true ,
8822   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8823 }
8824
8825
8826 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8827 {
8828   \IfPackageLoadedTF { tikz }

```



```

8829 {
8830   \group_begin:
8831   \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.

```

8832   \tl_set:Nn \l_tmpa_tl { { #2 } }
8833   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8834   { \@@_for_a_block:nnnnn ##1 }
8835   \@@_all_the_cells:
8836   \group_end:
8837 }
8838 { \@@_error:n { TikzEveryCell~without~tikz } }
8839 }
8840
8841 \tl_new:N \@@_i_tl
8842 \tl_new:N \@@_j_tl
8843
8844 \cs_new_protected:Nn \@@_all_the_cells:
8845 {
8846   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8847   {
8848     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8849     {
8850       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8851       {
8852         \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8853         { \@@_i_tl - \@@_j_tl }
8854         {
8855           \bool_set_false:N \l_tmpa_bool
8856           \cs_if_exist:cTF
8857           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8858           {
8859             \bool_if:NF \l_@@_empty_bool
8860             { \bool_set_true:N \l_tmpa_bool }
8861           }
8862           {
8863             \bool_if:NF \l_@@_not_empty_bool
8864             { \bool_set_true:N \l_tmpa_bool }
8865           }
8866           \bool_if:NT \l_tmpa_bool
8867           {
8868             \@@_block_tikz:nnnnV
8869             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8870           }
8871         }
8872       }
8873     }
8874   }
8875 }
8876
8877 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8878 {
8879   \bool_if:NF \l_@@_empty_bool
8880   {
8881     \@@_block_tikz:nnnnV
8882     { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8883   }
8884   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8885 }
8886
8887 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8888 {
8889   \int_step_inline:nnn { #1 } { #3 }

```

```

8890 {
8891   \int_step_inline:nnn { #2 } { #4 }
8892   { \cs_set:cpn { cell - ##1 - #####1 } { } }
8893 }
8894 }

```

37 The command \ShowCellNames

```

8895 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8896 {
8897   \dim_zero_new:N \g_@@_tmpc_dim
8898   \dim_zero_new:N \g_@@_tmpd_dim
8899   \dim_zero_new:N \g_@@_tmpe_dim
8900   \int_step_inline:nn \c@iRow
8901   {
8902     \begin { pgfpicture }
8903     \@@_qpoint:n { row - ##1 }
8904     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8905     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8906     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8907     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8908     \bool_if:NTF \l_@@_in_code_after_bool
8909     \end { pgfpicture }
8910     \int_step_inline:nn \c@jCol
8911     {
8912       \hbox_set:Nn \l_tmpa_box
8913       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8914       \begin { pgfpicture }
8915       \@@_qpoint:n { col - #####1 }
8916       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8917       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8918       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8919       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8920       \endpgfpicture
8921       \end { pgfpicture }
8922       \fp_set:Nn \l_tmpa_fp
8923       {
8924         \fp_min:nn
8925         {
8926           \fp_min:nn
8927           {
8928             \dim_ratio:nn
8929             { \g_@@_tmpd_dim }
8930             { \box_wd:N \l_tmpa_box }
8931           }
8932           {
8933             \dim_ratio:nn
8934             { \g_tmpb_dim }
8935             { \box_ht_plus_dp:N \l_tmpa_box }
8936           }
8937         }
8938         { 1.0 }
8939       }
8940       \box_scale:Nnn \l_tmpa_box
8941       { \fp_use:N \l_tmpa_fp }
8942       { \fp_use:N \l_tmpa_fp }
8943       \pgfpicture
8944       \pgfrememberpicturerepositiononpagetrue
8945       \pgf@relevantforpicturesizefalse
8946       \pgftransformshift
8947       {
8948         \pgfpoint

```

```

8949         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8950         { \dim_use:N \g_tmpa_dim }
8951     }
8952     \pgfnode
8953     { rectangle }
8954     { center }
8955     { \box_use:N \l_tmpa_box }
8956     { }
8957     { }
8958     \endpgfpicture
8959 }
8960 }
8961 }
8962 \NewDocumentCommand \@@_ShowCellNames { }
8963 {
8964     \bool_if:NT \l_@@_in_code_after_bool
8965     {
8966         \pgfpicture
8967         \pgfrememberpicturepositiononpagetrue
8968         \pgf@relevantforpicturesizefalse
8969         \pgfpathrectanglecorners
8970         { \@@_qpoint:n { 1 } }
8971         {
8972             \@@_qpoint:n
8973             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8974         }
8975         \pgfsetfillopacity { 0.75 }
8976         \pgfsetfillcolor { white }
8977         \pgfusepathqfill
8978         \endpgfpicture
8979     }
8980     \dim_zero_new:N \g_@@_tmpc_dim
8981     \dim_zero_new:N \g_@@_tmpd_dim
8982     \dim_zero_new:N \g_@@_tmpe_dim
8983     \int_step_inline:nn \c@iRow
8984     {
8985         \bool_if:NTF \l_@@_in_code_after_bool
8986         {
8987             \pgfpicture
8988             \pgfrememberpicturepositiononpagetrue
8989             \pgf@relevantforpicturesizefalse
8990         }
8991         { \begin { pgfpicture } }
8992         \@@_qpoint:n { row - ##1 }
8993         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8994         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8995         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8996         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8997         \bool_if:NTF \l_@@_in_code_after_bool
8998         { \endpgfpicture }
8999         { \end { pgfpicture } }
9000         \int_step_inline:nn \c@jCol
9001         {
9002             \hbox_set:Nn \l_tmpa_box
9003             {
9004                 \normalfont \Large \sffamily \bfseries
9005                 \bool_if:NTF \l_@@_in_code_after_bool
9006                 { \color { red } }
9007                 { \color { red ! 50 } }
9008                 ##1 - ###1
9009             }
9010             \bool_if:NTF \l_@@_in_code_after_bool
9011             {

```

```

9012         \pgfpicture
9013         \pgfrememberpicturerepositiononpagetrue
9014         \pgf@relevantforpicturesizefalse
9015     }
9016     { \begin { pgfpicture } }
9017     @@_qpoint:n { col - ####1 }
9018     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9019     @@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9020     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9021     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9022     \bool_if:NTF \l_@@_in_code_after_bool
9023     { \endpgfpicture }
9024     { \end { pgfpicture } }
9025     \fp_set:Nn \l_tmpa_fp
9026     {
9027         \fp_min:nn
9028         {
9029             \fp_min:nn
9030             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9031             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9032         }
9033         { 1.0 }
9034     }
9035     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9036     \pgfpicture
9037     \pgfrememberpicturerepositiononpagetrue
9038     \pgf@relevantforpicturesizefalse
9039     \pgftransformshift
9040     {
9041         \pgfpoint
9042         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9043         { \dim_use:N \g_tmpa_dim }
9044     }
9045     \pgfnode
9046     { rectangle }
9047     { center }
9048     { \box_use:N \l_tmpa_box }
9049     { }
9050     { }
9051     \endpgfpicture
9052 }
9053 }
9054 }

```

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.

```

9055 \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.

```

9056 \bool_new:N \g_@@_footnote_bool
9057 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9058 {
9059     The~key~'\l_keys_key_str'~is~unknown. \

```

```

9060     That~key~will~be~ignored. \\
9061     For~a~list~of~the~available~keys,~type~H~<return>.
9062   }
9063   {
9064     The~available~keys~are~(in~alphabetic~order):~
9065     footnote,~
9066     footnotehyper,~
9067     messages~for~Overleaf,~
9068     no~test~for~array,~
9069     renew~dots,~and~
9070     renew~matrix.
9071   }
9072   \keys_define:nn { NiceMatrix / Package }
9073   {
9074     renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9075     renew~dots .value_forbidden:n = true ,
9076     renew~matrix .code:n = \@@_renew_matrix: ,
9077     renew~matrix .value_forbidden:n = true ,
9078     messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9079     footnote .bool_set:N = \g_@@_footnote_bool ,
9080     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9081     no~test~for~array .bool_set:N = \g_@@_no_test_for_array_bool ,
9082     no~test~for~array .default:n = true ,
9083     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9084   }
9085   \ProcessKeysOptions { NiceMatrix / Package }

9086   \@@_msg_new:nn { footnote~with~footnotehyper~package }
9087   {
9088     You~can't~use~the~option~'footnote'~because~the~package~
9089     footnotehyper~has~already~been~loaded.~
9090     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9091     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9092     of~the~package~footnotehyper.\\
9093     The~package~footnote~won't~be~loaded.
9094   }

9095   \@@_msg_new:nn { footnotehyper~with~footnote~package }
9096   {
9097     You~can't~use~the~option~'footnotehyper'~because~the~package~
9098     footnote~has~already~been~loaded.~
9099     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9100     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9101     of~the~package~footnote.\\
9102     The~package~footnotehyper~won't~be~loaded.
9103   }

9104   \bool_if:NT \g_@@_footnote_bool
9105   {

The class beamer has its own system to extract footnotes and that's why we have nothing to do if
beamer is used.

9106     \IfClassLoadedTF { beamer }
9107     { \bool_set_false:N \g_@@_footnote_bool }
9108     {
9109       \IfPackageLoadedTF { footnotehyper }
9110       { \@@_error:n { footnote~with~footnotehyper~package } }
9111       { \usepackage { footnote } }
9112     }
9113   }

9114   \bool_if:NT \g_@@_footnotehyper_bool
9115   {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9116 \IfClassLoadedTF { beamer }
9117 { \bool_set_false:N \g_@@_footnote_bool }
9118 {
9119   \IfPackageLoadedTF { footnote }
9120   { \@@_error:n { footnotehyper~with~footnote~package } }
9121   { \usepackage { footnotehyper } }
9122 }
9123 \bool_set_true:N \g_@@_footnote_bool
9124 }

```

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.

```

9125 \bool_new:N \l_@@_underscore_loaded_bool
9126 \IfPackageLoadedTF { underscore }
9127 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9128 { }

9129 \hook_gput_code:nnn { begindocument } { . }
9130 {
9131   \bool_if:NF \l_@@_underscore_loaded_bool
9132   {
9133     \IfPackageLoadedTF { underscore }
9134     { \@@_error:n { underscore~after~nicematrix } }
9135     { }
9136   }
9137 }

```

40 Error messages of the package

```

9138 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9139 { \str_const:Nn \c_@@_available_keys_str { } }
9140 {
9141   \str_const:Nn \c_@@_available_keys_str
9142   { For~a~list~of~the~available~keys,~type~H~<return>. }
9143 }

9144 \seq_new:N \g_@@_types_of_matrix_seq
9145 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9146 {
9147   NiceMatrix ,
9148   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9149 }
9150 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9151 { \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:NoTF` 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`.

```

9152 \cs_new_protected:Npn \@@_error_too_much_cols:
9153 {
9154   \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9155   {
9156     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9157     { \@@_fatal:n { too-much-cols-for-matrix } }
9158     {
9159       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9160       { \@@_fatal:n { too-much-cols-for-matrix } }
9161       {
9162         \bool_if:NF \l_@@_last_col_without_value_bool
9163         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9164       }
9165     }
9166   }
9167   { \@@_fatal:nn { too-much-cols-for-array } }
9168 }

```

The following command must *not* be protected since it's used in an error message.

```

9169 \cs_new:Npn \@@_message_hdotsfor:
9170 {
9171   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9172   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
9173 }
9174 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9175 {
9176   Incompatible-options.\\
9177   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9178   The-output-will-not-be-reliable.
9179 }
9180 \@@_msg_new:nn { negative-weight }
9181 {
9182   Negative-weight.\\
9183   The-weight-of~the~'X'~columns~must~be~positive~and~you~have~used~
9184   the~value~'\int_use:N \l_@@_weight_int'.\\
9185   The~absolute~value~will~be~used.
9186 }
9187 \@@_msg_new:nn { last-col-not-used }
9188 {
9189   Column~not~used.\\
9190   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9191   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9192 }
9193 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9194 {
9195   Too-much-columns.\\
9196   In~the~row~\int_eval:n { \c@iRow },~
9197   you~try~to~use~more~columns~
9198   than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\\
9199   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9200   (plus~the~exterior~columns).~This~error~is~fatal.
9201 }
9202 \@@_msg_new:nn { too-much-cols-for-matrix }
9203 {
9204   Too-much-columns.\\
9205   In~the~row~\int_eval:n { \c@iRow },~
9206   you~try~to~use~more~columns~than~allowed~by~your~
9207   \@@_full_name_env:.~\@@_message_hdotsfor:\\ Recall~that~the~maximal~
9208   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9209   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~

```

```

9210     Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9211     \token_to_str:N \setcounter\ to~change~that~value).~
9212     This-error-is-fatal.
9213 }

9214 \@@_msg_new:nn { too-much-cols-for-array }
9215 {
9216     Too-much-columns.\\
9217     In~the~row~\int_eval:n { \c@iRow },~
9218     ~you~try~to~use~more~columns~than~allowed~by~your~
9219     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9220     \int_use:N \g_@@_static_num_of_col_int\
9221     ~(plus~the~potential~exterior~ones).
9222     This-error-is-fatal.
9223 }

9224 \@@_msg_new:nn { columns-not-used }
9225 {
9226     Columns-not-used.\\
9227     The-preamble-of~your~\@@_full_name_env:\ announces~\int_use:N
9228     \g_@@_static_num_of_col_int\ columns-but~you~use~only~\int_use:N \c@jCol.\\
9229     The-columns-you-did-not-used-won't-be-created.\\
9230     You-won't-have~similar-error~till~the-end-of~the~document.
9231 }

9232 \@@_msg_new:nn { in-first-col }
9233 {
9234     Erroneous-use.\\
9235     You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9236     That-command-will-be-ignored.
9237 }

9238 \@@_msg_new:nn { in-last-col }
9239 {
9240     Erroneous-use.\\
9241     You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9242     That-command-will-be-ignored.
9243 }

9244 \@@_msg_new:nn { in-first-row }
9245 {
9246     Erroneous-use.\\
9247     You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9248     That-command-will-be-ignored.
9249 }

9250 \@@_msg_new:nn { in-last-row }
9251 {
9252     You-can't-use-the-command~#1 in-the-last-row~(exterior)~of~the-array.\\
9253     That-command-will-be-ignored.
9254 }

9255 \@@_msg_new:nn { caption-outside-float }
9256 {
9257     Key-caption~forbidden.\\
9258     You-can't-use-the-key~'caption'~because~you~are~not~in~a~floating~
9259     environment.~This-key-will-be-ignored.
9260 }

9261 \@@_msg_new:nn { short-caption-without-caption }
9262 {
9263     You-should-not-use-the-key~'short-caption'~without~'caption'.~
9264     However,~your~'short-caption'~will~be~used~as~'caption'.
9265 }

9266 \@@_msg_new:nn { double-closing-delimiter }
9267 {
9268     Double~delimiter.\\

```



```

9269     You-can't-put-a-second-closing-delimiter-~"#1"~just-after-a-first-closing~
9270     delimiter.~This-delimiter-will-be-ignored.
9271 }
9272 \@@_msg_new:nn { delimiter-after-opening }
9273 {
9274     Double-delimiter.\\
9275     You-can't-put-a-second-delimiter-~"#1"~just-after-a-first-opening~
9276     delimiter.~That-delimiter-will-be-ignored.
9277 }
9278 \@@_msg_new:nn { bad-option-for-line-style }
9279 {
9280     Bad-line-style.\\
9281     Since-you-haven't-loaded-Tikz,~the-only~value-you-can-give-to-'line-style'~
9282     is~'standard'.~That-key-will-be-ignored.
9283 }
9284 \@@_msg_new:nn { Identical-notes-in-caption }
9285 {
9286     Identical-tabular-notes.\\
9287     You-can't-put-several-notes-with-the-same-content-in~
9288     \token_to_str:N \caption\ (but-you-can-in-the-main-tabular).\\
9289     If-you-go-on,~the-output-will-probably-be-erroneous.
9290 }
9291 \@@_msg_new:nn { tabularnote-below-the-tabular }
9292 {
9293     \token_to_str:N \tabularnote\ forbidden\\
9294     You-can't-use-\token_to_str:N \tabularnote\ in-the-caption~
9295     of-your-tabular-because-the-caption-will-be-composed-below~
9296     the-tabular.~If-you-want-the-caption-above-the-tabular~use-the~
9297     key-'caption-above'~in-\token_to_str:N \NiceMatrixOptions.\\
9298     Your-\token_to_str:N \tabularnote\ will-be-discarded-and~
9299     no-similar-error-will-be-raised-in-this-document.
9300 }
9301 \@@_msg_new:nn { Unknown-key-for-rules }
9302 {
9303     Unknown-key.\\
9304     There-is-only-two-keys-available-here:~width-and-color.\\
9305     Your-key-'~\l_keys_key_str'~will-be-ignored.
9306 }
9307 \@@_msg_new:nn { Unknown-key-for-TikzEveryCell }
9308 {
9309     Unknown-key.\\
9310     There-is-only-two-keys-available-here:~
9311     'empty'~and~'not-empty'.\\
9312     Your-key-'~\l_keys_key_str'~will-be-ignored.
9313 }
9314 \@@_msg_new:nn { Unknown-key-for-rotate }
9315 {
9316     Unknown-key.\\
9317     The-only-key-available-here-is~'c'.\\
9318     Your-key-'~\l_keys_key_str'~will-be-ignored.
9319 }
9320 \@@_msg_new:nnn { Unknown-key-for-custom-line }
9321 {
9322     Unknown-key.\\
9323     The-key-'~\l_keys_key_str'~is-unknown-in-a-'custom-line'.~
9324     It-you-go-on,~you-will-probably-have-other-errors. \\
9325     \c_@@_available_keys_str
9326 }
9327 {
9328     The-available-keys-are~(in~alphabetic~order):~

```

```

9329     ccommand,~
9330     color,~
9331     command,~
9332     dotted,~
9333     letter,~
9334     multiplicity,~
9335     sep-color,~
9336     tikz,~and~total-width.
9337 }
9338 \@@_msg_new:nnn { Unknown~key~for~xdots }
9339 {
9340   Unknown~key.\\
9341   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9342   \c_@@_available_keys_str
9343 }
9344 {
9345   The~available~keys~are~(in~alphabetic~order):~
9346   'color',~
9347   'horizontal-labels',~
9348   'inter',~
9349   'line-style',~
9350   'radius',~
9351   'shorten',~
9352   'shorten-end'~and~'shorten-start'.
9353 }
9354 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9355 {
9356   Unknown~key.\\
9357   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9358   (and~you~try~to~use~'\l_keys_key_str')\\
9359   That~key~will~be~ignored.
9360 }
9361 \@@_msg_new:nn { label~without~caption }
9362 {
9363   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9364   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9365 }
9366 \@@_msg_new:nn { W~warning }
9367 {
9368   Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9369   (row~\int_use:N \c@iRow).
9370 }
9371 \@@_msg_new:nn { Construct~too~large }
9372 {
9373   Construct~too~large.\\
9374   Your~command~\token_to_str:N #1
9375   can't~be~drawn~because~your~matrix~is~too~small.\\
9376   That~command~will~be~ignored.
9377 }
9378 \@@_msg_new:nn { underscore~after~nicematrix }
9379 {
9380   Problem~with~'underscore'.\\
9381   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9382   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9383   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}}'.
9384 }
9385 \@@_msg_new:nn { ampersand~in~light-syntax }
9386 {
9387   Ampersand~forbidden.\\
9388   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9389   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.

```

```

9390 }
9391 \@@_msg_new:nn { double-backslash~in~light-syntax }
9392 {
9393   Double~backslash~forbidden.\\
9394   You~can't~use~\token_to_str:N
9395   \\~to~separate~rows~because~the~key~'light-syntax'~
9396   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9397   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9398 }
9399 \@@_msg_new:nn { hlines~with~color }
9400 {
9401   Incompatible~keys.\\
9402   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9403   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9404   Maybe~it~will~possible~in~future~version.\\
9405   Your~key~will~be~discarded.
9406 }
9407 \@@_msg_new:nn { bad~value~for~baseline }
9408 {
9409   Bad~value~for~baseline.\\
9410   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9411   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9412   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9413   the~form~'line-i'.\\
9414   A~value~of~1~will~be~used.
9415 }
9416 \@@_msg_new:nn { detection~of~empty~cells }
9417 {
9418   Problem~with~'not-empty'\\
9419   For~technical~reasons,~you~must~activate~
9420   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9421   in~order~to~use~the~key~'\l_keys_key_str'.\\
9422   That~key~will~be~ignored.
9423 }
9424 \@@_msg_new:nn { siunitx~not~loaded }
9425 {
9426   siunitx~not~loaded\\
9427   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9428   That~error~is~fatal.
9429 }
9430 \@@_msg_new:nn { ragged2e~not~loaded }
9431 {
9432   You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9433   your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9434   \l_keys_key_str'~will~be~used~instead.
9435 }
9436 \@@_msg_new:nn { Invalid~name }
9437 {
9438   Invalid~name.\\
9439   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9440   \SubMatrix\ of~your~\@@_full_name_env:.\
9441   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9442   This~key~will~be~ignored.
9443 }
9444 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9445 {
9446   Wrong~line.\\
9447   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9448   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9449   number~is~not~valid.~It~will~be~ignored.

```

```

9450 }
9451 \@@_msg_new:nn { Impossible-delimiter }
9452 {
9453   Impossible-delimiter.\
9454   It's-impossible-to-draw-the-#1-delimiter-of-your-
9455   \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty-
9456   in-that-column.
9457   \bool_if:NT \l_@@_submatrix_slim_bool
9458   { ~Maybe-you-should-try-without-the-key-'slim'. } \
9459   This-\token_to_str:N \SubMatrix\ will-be-ignored.
9460 }
9461 \@@_msg_new:nnn { width-without-X-columns }
9462 {
9463   You-have-used-the-key-'width'~but-you-have-put-no-'X'~column.~
9464   That-key-will-be-ignored.
9465 }
9466 {
9467   This-message-is-the-message-'width-without-X-columns'~
9468   of-the-module-'nicematrix'.~
9469   The-experimented-users-can-disable-that-message-with-
9470   \token_to_str:N \msg_redirect_name:nnn.\
9471 }
9472
9473 \@@_msg_new:nn { key-multiplicity-with-dotted }
9474 {
9475   Incompatible-keys. \
9476   You-have-used-the-key-'multiplicity'~with-the-key-'dotted'~
9477   in-a-'custom-line'.~They-are-incompatible. \
9478   The-key-'multiplicity'~will-be-discarded.
9479 }
9480 \@@_msg_new:nn { empty-environment }
9481 {
9482   Empty-environment.\
9483   Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9484 }
9485 \@@_msg_new:nn { No-letter-and-no-command }
9486 {
9487   Erroneous-use.\
9488   Your-use-of-'custom-line'~is-no-op~since-you-don't-have-used-the-
9489   key-'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys-'command'~or~
9490   ~'ccommand'~(to-draw-horizontal-rules).\
9491   However,~you~can~go~on.
9492 }
9493 \@@_msg_new:nn { Forbidden-letter }
9494 {
9495   Forbidden-letter.\
9496   You-can't-use-the-letter~'#1'~for-a-customized-line.\
9497   It~will~be~ignored.
9498 }
9499 \@@_msg_new:nn { Several-letters }
9500 {
9501   Wrong-name.\
9502   You-must-use-only-one-letter~as-value-for-the-key-'letter'~(and-you-
9503   have-used~'\l_@@_letter_str').\
9504   It~will~be~ignored.
9505 }
9506 \@@_msg_new:nn { Delimiter-with-small }
9507 {
9508   Delimiter~forbidden.\
9509   You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\

```

```

9510     because~the~key~'small'~is~in~force.\\
9511     This~error~is~fatal.
9512 }
9513 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9514 {
9515     Unknown~cell.\\
9516     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9517     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9518     can't~be~executed~because~a~cell~doesn't~exist.\\
9519     This~command~\token_to_str:N \line\ will~be~ignored.
9520 }
9521 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9522 {
9523     Duplicate~name.\\
9524     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9525     in~this~\@@_full_name_env:.\\
9526     This~key~will~be~ignored.\\
9527     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9528     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9529 }
9530 {
9531     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9532     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9533 }
9534 \@@_msg_new:nn { r~or~l~with~preamble }
9535 {
9536     Erroneous~use.\\
9537     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9538     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9539     your~\@@_full_name_env:.\\
9540     This~key~will~be~ignored.
9541 }
9542 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9543 {
9544     Erroneous~use.\\
9545     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9546     the~array.~This~error~is~fatal.
9547 }
9548 \@@_msg_new:nn { bad~corner }
9549 {
9550     Bad~corner.\\
9551     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9552     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9553     This~specification~of~corner~will~be~ignored.
9554 }
9555 \@@_msg_new:nn { bad~border }
9556 {
9557     Bad~border.\\
9558     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9559     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9560     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9561     also~use~the~key~'tikz'
9562     \IfPackageLoadedTF { tikz }
9563     { }
9564     {~if~you~load~the~LaTeX~package~'tikz'}).\\
9565     This~specification~of~border~will~be~ignored.
9566 }
9567 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9568 {
9569     TikZ~not~loaded.\\
9570     You~can't~use~\token_to_str:N \TikzEveryCell\

```

```

9571     because-you-have-not-loaded-tikz.~
9572     This-command-will-be-ignored.
9573 }
9574 \@@_msg_new:nn { tikz-key-without-tikz }
9575 {
9576     TikZ-not-loaded.\\
9577     You-can't-use-the-key-'tikz'~for-the-command~'\token_to_str:N
9578     \Block'~because-you-have-not-loaded-tikz.~
9579     This-key-will-be-ignored.
9580 }
9581 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
9582 {
9583     Erroneous-use.\\
9584     In-the~\@@_full_name_env:,~you-must-use-the-key~
9585     'last-col'~without-value.\\
9586     However,~you-can-go-on-for-this-time~
9587     (the-value~'\l_keys_value_tl'~will-be-ignored).
9588 }
9589 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9590 {
9591     Erroneous-use.\\
9592     In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9593     'last-col'~without-value.\\
9594     However,~you-can-go-on-for-this-time~
9595     (the-value~'\l_keys_value_tl'~will-be-ignored).
9596 }
9597 \@@_msg_new:nn { Block-too-large-1 }
9598 {
9599     Block-too-large.\\
9600     You-try-to-draw-a-block-in-the-cell~#1-#2-of-your-matrix-but-the-matrix-is-
9601     too-small-for-that-block. \\
9602     This-block-and-maybe-others-will-be-ignored.
9603 }
9604 \@@_msg_new:nn { Block-too-large-2 }
9605 {
9606     Block-too-large.\\
9607     The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9608     \g_@@_static_num_of_col_int\
9609     columns-but-you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
9610     specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9611     (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\\
9612     This-block-and-maybe-others-will-be-ignored.
9613 }
9614 \@@_msg_new:nn { unknown-column-type }
9615 {
9616     Bad-column-type.\\
9617     The-column-type~'#1'~in-your~\@@_full_name_env:\
9618     is-unknown. \\
9619     This-error-is-fatal.
9620 }
9621 \@@_msg_new:nn { unknown-column-type-S }
9622 {
9623     Bad-column-type.\\
9624     The-column-type~'S'~in-your~\@@_full_name_env:\ is-unknown. \\
9625     If-you-want-to-use-the-column-type~'S'~of-siunitx,~you-should-
9626     load-that-package. \\
9627     This-error-is-fatal.
9628 }
9629 \@@_msg_new:nn { tabularnote-forbidden }
9630 {

```

```

9631     Forbidden-command.\
9632     You-can't-use-the-command~\token_to_str:N\tabularnote\
9633     ~here~This-command-is-available-only-in~
9634     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9635     the-argument-of-a-command~\token_to_str:N \caption\ included~
9636     in-an-environment~{table}. \
9637     This-command-will-be-ignored.
9638   }

9639   \@@_msg_new:nn { borders~forbidden }
9640   {
9641     Forbidden-key.\
9642     You-can't-use-the-key~'borders'~of-the-command~\token_to_str:N \Block\
9643     because-the-option~'rounded-corners'~
9644     is-in-force-with-a-non-zero-value.\
9645     This-key-will-be-ignored.
9646   }

9647   \@@_msg_new:nn { bottomrule~without~booktabs }
9648   {
9649     booktabs-not-loaded.\
9650     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
9651     loaded~'booktabs'.\
9652     This-key-will-be-ignored.
9653   }

9654   \@@_msg_new:nn { enumitem~not~loaded }
9655   {
9656     enumitem-not-loaded.\
9657     You-can't-use-the-command~\token_to_str:N\tabularnote\
9658     ~because-you-haven't~loaded~'enumitem'.\
9659     All-the-commands~\token_to_str:N\tabularnote\ will-be~
9660     ignored-in-the-document.
9661   }

9662   \@@_msg_new:nn { tikz~without~tikz }
9663   {
9664     Tikz-not-loaded.\
9665     You-can't-use-the-key~'tikz'~here~because-Tikz-is-not~
9666     loaded~If-you-go-on,~that-key-will-be-ignored.
9667   }

9668   \@@_msg_new:nn { tikz-in~custom-line~without~tikz }
9669   {
9670     Tikz-not-loaded.\
9671     You-have-used-the-key~'tikz'~in-the-definition-of-a~
9672     customized-line~(with~'custom-line')~but~tikz-is-not-loaded.~
9673     You-can-go-on-but-you-will-have-another-error-if-you-actually~
9674     use-that-custom-line.
9675   }

9676   \@@_msg_new:nn { tikz-in~borders~without~tikz }
9677   {
9678     Tikz-not-loaded.\
9679     You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9680     command~\token_to_str:N\Block')~but~tikz-is-not-loaded.~
9681     That-key-will-be-ignored.
9682   }

9683   \@@_msg_new:nn { without~color~inside }
9684   {
9685     If-order-to-use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9686     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9687     outside~\token_to_str:N \CodeBefore,~you~
9688     should~have~used~the-key~'color~inside'~in-your~\@@_full_name_env:.\
9689     You-can-go-on-but-you-may~need~more~compilations.
9690   }

```

```

9691 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9692 {
9693   Erroneous-use.\
9694   In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
9695   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9696   The~key~'color'~will~be~discarded.
9697 }
9698 \@@_msg_new:nn { Wrong-last-row }
9699 {
9700   Wrong-number.\
9701   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9702   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9703   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9704   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9705   without~value~(more~compilations~might~be~necessary).
9706 }
9707 \@@_msg_new:nn { Yet-in-env }
9708 {
9709   Nested-environments.\
9710   Environments-of~nicematrix~can't~be~nested.\
9711   This~error~is~fatal.
9712 }
9713 \@@_msg_new:nn { Outside-math-mode }
9714 {
9715   Outside-math-mode.\
9716   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9717   (and~not~in~\token_to_str:N \vcenter).\
9718   This~error~is~fatal.
9719 }
9720 \@@_msg_new:nn { One-letter-allowed }
9721 {
9722   Bad-name.\
9723   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
9724   It~will~be~ignored.
9725 }
9726 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9727 {
9728   Environment~{TabularNote}~forbidden.\
9729   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9730   but~*before*~the~\token_to_str:N \CodeAfter.\
9731   This~environment~{TabularNote}~will~be~ignored.
9732 }
9733 \@@_msg_new:nn { varwidth-not-loaded }
9734 {
9735   varwidth-not-loaded.\
9736   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9737   loaded.\
9738   Your~column~will~behave~like~'p'.
9739 }
9740 \@@_msg_new:nnn { Unknow-key-for-RulesBis }
9741 {
9742   Unknow-key.\
9743   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
9744   \c_@@_available_keys_str
9745 }
9746 {
9747   The~available~keys~are~(in~alphabetic~order):~
9748   color,~
9749   dotted,~
9750   multiplicity,~
9751   sep-color,~

```



```

9752     tikz,~and~total~width.
9753 }
9754
9755 \@@_msg_new:nnn { Unknown~key~for~Block }
9756 {
9757     Unknown~key.\\
9758     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9759     \Block.\\ It~will~be~ignored. \\
9760     \c_@@_available_keys_str
9761 }
9762 {
9763     The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9764     hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9765     respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9766 }
9767 \@@_msg_new:nnn { Unknown~key~for~Brace }
9768 {
9769     Unknown~key.\\
9770     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9771     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9772     It~will~be~ignored. \\
9773     \c_@@_available_keys_str
9774 }
9775 {
9776     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9777     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9778     right~shorten)~and~yshift.
9779 }
9780 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9781 {
9782     Unknown~key.\\
9783     The~key~'\l_keys_key_str'~is~unknown.\\
9784     It~will~be~ignored. \\
9785     \c_@@_available_keys_str
9786 }
9787 {
9788     The~available~keys~are~(in~alphabetic~order):~
9789     delimiters/color,~
9790     rules~(with~the~subkeys~'color'~and~'width'),~
9791     sub~matrix~(several~subkeys)~
9792     and~xdots~(several~subkeys).~
9793     The~latter~is~for~the~command~\token_to_str:N \line.
9794 }
9795 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9796 {
9797     Unknown~key.\\
9798     The~key~'\l_keys_key_str'~is~unknown.\\
9799     It~will~be~ignored. \\
9800     \c_@@_available_keys_str
9801 }
9802 {
9803     The~available~keys~are~(in~alphabetic~order):~
9804     create~cell~nodes,~
9805     delimiters/color~and~
9806     sub~matrix~(several~subkeys).
9807 }
9808 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9809 {
9810     Unknown~key.\\
9811     The~key~'\l_keys_key_str'~is~unknown.\\
9812     That~key~will~be~ignored. \\
9813     \c_@@_available_keys_str

```

```

9814 }
9815 {
9816   The-available-keys-are~(in~alphabetic~order):~
9817   'delimiters/color',~
9818   'extra-height',~
9819   'hlines',~
9820   'hvlines',~
9821   'left-xshift',~
9822   'name',~
9823   'right-xshift',~
9824   'rules'~(with~the~subkeys~'color'~and~'width'),~
9825   'slim',~
9826   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9827   and~'right-xshift').\\
9828 }
9829 \@@_msg_new:nnn { Unknown~key~for~notes }
9830 {
9831   Unknown-key.\\
9832   The-key~'\l_keys_key_str'~is~unknown.\\
9833   That~key~will~be~ignored. \\
9834   \c_@@_available_keys_str
9835 }
9836 {
9837   The-available-keys-are~(in~alphabetic~order):~
9838   bottomrule,~
9839   code-after,~
9840   code-before,~
9841   detect-duplicates,~
9842   enumitem-keys,~
9843   enumitem-keys-para,~
9844   para,~
9845   label-in-list,~
9846   label-in-tabular~and~
9847   style.
9848 }
9849 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9850 {
9851   Unknown-key.\\
9852   The-key~'\l_keys_key_str'~is~unknown~for~the~command~
9853   \token_to_str:N \RowStyle. \\
9854   That~key~will~be~ignored. \\
9855   \c_@@_available_keys_str
9856 }
9857 {
9858   The-available-keys-are~(in~alphabetic~order):~
9859   'bold',~
9860   'cell-space-top-limit',~
9861   'cell-space-bottom-limit',~
9862   'cell-space-limits',~
9863   'color',~
9864   'nb-rows'~and~
9865   'rowcolor'.
9866 }
9867 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9868 {
9869   Unknown-key.\\
9870   The-key~'\l_keys_key_str'~is~unknown~for~the~command~
9871   \token_to_str:N \NiceMatrixOptions. \\
9872   That~key~will~be~ignored. \\
9873   \c_@@_available_keys_str
9874 }
9875 {
9876   The-available-keys-are~(in~alphabetic~order):~

```

```

9877 allow-duplicate-names,~
9878 caption-above,~
9879 cell-space-bottom-limit,~
9880 cell-space-limits,~
9881 cell-space-top-limit,~
9882 code-for-first-col,~
9883 code-for-first-row,~
9884 code-for-last-col,~
9885 code-for-last-row,~
9886 corners,~
9887 custom-key,~
9888 create-extra-nodes,~
9889 create-medium-nodes,~
9890 create-large-nodes,~
9891 delimiters~(several~subkeys),~
9892 end-of-row,~
9893 first-col,~
9894 first-row,~
9895 hlines,~
9896 hvlines,~
9897 hvlines-except-borders,~
9898 last-col,~
9899 last-row,~
9900 left-margin,~
9901 light-syntax,~
9902 matrix/columns-type,~
9903 notes~(several~subkeys),~
9904 nullify-dots,~
9905 pgf-node-code,~
9906 renew-dots,~
9907 renew-matrix,~
9908 respect-arraystretch,~
9909 rounded-corners,~
9910 right-margin,~
9911 rules~(with~the~subkeys~'color'~and~'width'),~
9912 small,~
9913 sub-matrix~(several~subkeys),~
9914 vlines,~
9915 xdots~(several~subkeys).
9916 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9917 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9918 {
9919   Unknown~key.\\
9920   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9921   \{NiceArray\}. \\
9922   That~key~will~be~ignored. \\
9923   \c_@@_available_keys_str
9924 }
9925 {
9926   The~available~keys~are~(in~alphabetic~order):~
9927   b,~
9928   baseline,~
9929   c,~
9930   cell-space-bottom-limit,~
9931   cell-space-limits,~
9932   cell-space-top-limit,~
9933   code-after,~
9934   code-for-first-col,~
9935   code-for-first-row,~
9936   code-for-last-col,~
9937   code-for-last-row,~

```

```

9938     color-inside,~
9939     columns-width,~
9940     corners,~
9941     create-extra-nodes,~
9942     create-medium-nodes,~
9943     create-large-nodes,~
9944     extra-left-margin,~
9945     extra-right-margin,~
9946     first-col,~
9947     first-row,~
9948     hlines,~
9949     hvlines,~
9950     hvlines-except-borders,~
9951     last-col,~
9952     last-row,~
9953     left-margin,~
9954     light-syntax,~
9955     name,~
9956     nullify-dots,~
9957     pgf-node-code,~
9958     renew-dots,~
9959     respect-arraystretch,~
9960     right-margin,~
9961     rounded-corners,~
9962     rules~(with~the~subkeys~'color'~and~'width'),~
9963     small,~
9964     t,~
9965     vlines,~
9966     xdots/color,~
9967     xdots/shorten-start,~
9968     xdots/shorten-end,~
9969     xdots/shorten-and~
9970     xdots/line-style.
9971 }

```

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).

```

9972 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9973 {
9974   Unknown~key.\\
9975   The~key~'\l_keys_key_str'~is~unknown~for~the~
9976   \@@_full_name_env:. \\
9977   That~key~will~be~ignored. \\
9978   \c_@@_available_keys_str
9979 }
9980 {
9981   The~available~keys~are~(in~alphabetic~order):~
9982   b,~
9983   baseline,~
9984   c,~
9985   cell-space-bottom-limit,~
9986   cell-space-limits,~
9987   cell-space-top-limit,~
9988   code-after,~
9989   code-for-first-col,~
9990   code-for-first-row,~
9991   code-for-last-col,~
9992   code-for-last-row,~
9993   color-inside,~
9994   columns-type,~
9995   columns-width,~
9996   corners,~
9997   create-extra-nodes,~
9998   create-medium-nodes,~

```

```

9999     create-large-nodes,~
10000     extra-left-margin,~
10001     extra-right-margin,~
10002     first-col,~
10003     first-row,~
10004     hlines,~
10005     hvlines,~
10006     hvlines-except-borders,~
10007     l,~
10008     last-col,~
10009     last-row,~
10010     left-margin,~
10011     light-syntax,~
10012     name,~
10013     nullify-dots,~
10014     pgf-node-code,~
10015     r,~
10016     renew-dots,~
10017     respect-arraystretch,~
10018     right-margin,~
10019     rounded-corners,~
10020     rules~(with~the~subkeys~'color'~and~'width'),~
10021     small,~
10022     t,~
10023     vlines,~
10024     xdots/color,~
10025     xdots/shorten-start,~
10026     xdots/shorten-end,~
10027     xdots/shorten-and~
10028     xdots/line-style.
10029 }
10030 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10031 {
10032     Unknown~key.\\
10033     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10034     \{NiceTabular\}. \\
10035     That~key~will~be~ignored. \\
10036     \c_@@_available_keys_str
10037 }
10038 {
10039     The~available~keys~are~(in~alphabetic~order):~
10040     b,~
10041     baseline,~
10042     c,~
10043     caption,~
10044     cell-space-bottom-limit,~
10045     cell-space-limits,~
10046     cell-space-top-limit,~
10047     code-after,~
10048     code-for-first-col,~
10049     code-for-first-row,~
10050     code-for-last-col,~
10051     code-for-last-row,~
10052     color-inside,~
10053     columns-width,~
10054     corners,~
10055     custom-line,~
10056     create-extra-nodes,~
10057     create-medium-nodes,~
10058     create-large-nodes,~
10059     extra-left-margin,~
10060     extra-right-margin,~
10061     first-col,~

```

```

10062 first-row,~
10063 hlines,~
10064 hvlines,~
10065 hvlines-except-borders,~
10066 label,~
10067 last-col,~
10068 last-row,~
10069 left-margin,~
10070 light-syntax,~
10071 name,~
10072 notes~(several~subkeys),~
10073 nullify-dots,~
10074 pgf-node-code,~
10075 renew-dots,~
10076 respect-arraystretch,~
10077 right-margin,~
10078 rounded-corners,~
10079 rules~(with~the~subkeys~'color'~and~'width'),~
10080 short-caption,~
10081 t,~
10082 tabularnote,~
10083 vlides,~
10084 xdots/color,~
10085 xdots/shorten-start,~
10086 xdots/shorten-end,~
10087 xdots/shorten-and~
10088 xdots/line-style.
10089 }

10090 \@@_msg_new:nnn { Duplicate-name }
10091 {
10092   Duplicate~name.\\
10093   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10094   the~same~environment~name~twice.~You~can~go~on,~but,~
10095   maybe,~you~will~have~incorrect~results~especially~
10096   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10097   message~again,~use~the~key~'allow-duplicate-names'~in~
10098   '\token_to_str:N \NiceMatrixOptions'.\\
10099   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10100     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10101 }
10102 {
10103   The~names~already~defined~in~this~document~are:~
10104   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10105 }

10106 \@@_msg_new:nn { Option-auto-for-columns-width }
10107 {
10108   Erroneous~use.\\
10109   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10110   That~key~will~be~ignored.
10111 }

10112 \@@_msg_new:nn { NiceTabularX~without~X }
10113 {
10114   NiceTabularX~without~X.\\
10115   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10116   However,~you~can~go~on.
10117 }

10118 \@@_msg_new:nn { Preamble~forgotten }
10119 {
10120   Preamble~forgotten.\\
10121   You~have~probably~forgotten~the~preamble~of~your~
10122   \@@_full_name_env:. \\
10123   This~error~is~fatal.

```


Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
3	Collecting options	4
4	Technical definitions	4
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>	36
10	The <code>\CodeBefore</code>	49
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	58
13	The redefinition of <code>\multicolumn</code>	73
14	The environment <code>{NiceMatrix}</code> and its variants	91
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
16	After the construction of the array	93
17	We draw the dotted lines	99
18	The actual instructions for drawing the dotted lines with Tikz	113
19	User commands available in the new environments	118
20	The command <code>\line</code> accessible in code-after	124
21	The command <code>\RowStyle</code>	126
22	Colors of cells, rows and columns	128
23	The vertical and horizontal rules	141
24	The empty corners	155
25	The environment <code>{NiceMatrixBlock}</code>	158
26	The extra nodes	159
27	The blocks	163
28	How to draw the dotted lines transparently	183
29	Automatic arrays	184
30	The redefinition of the command <code>\dotfill</code>	185

31	The command <code>\diagbox</code>	185
32	The keyword <code>\CodeAfter</code>	187
33	The delimiters in the preamble	188
34	The command <code>\SubMatrix</code>	189
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	197
36	The command <code>TikzEveryCell</code>	200
37	The command <code>\ShowCellNames</code>	202
38	We process the options at package loading	204
39	About the package underscore	206
40	Error messages of the package	206