

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

November 22, 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 6.26 of `nicematrix`, at the date of 2023/11/22.

```

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:Vn \c_sys_jobname_str { _region_ } % for Emacs
31   || \str_if_eq_p:Vn \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 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).

```

114 \tl_new:N \l_@@_argspec_tl
115 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
116 \cs_generate_variant:Nn \str_lowercase:n { V }

117 \hook_gput_code:nnn { begindocument } { . }
118 {
119   \IfPackageLoadedTF { tikz }
120   {

```

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.

```

121     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
122     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
123   }
124   {
125     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
126     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
127   }
128 }

```

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`).

```

129 \IfClassLoadedTF { revtex4-1 }
130 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
131 {
132   \IfClassLoadedTF { revtex4-2 }
133   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
134   {

```

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

```

135     \cs_if_exist:NT \rvtx@ifformat@geq
136     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
137     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
138   }
139 }

```

```

140 \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.

```

141 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
142 {
143   \iow_now:Nn \@mainaux
144   {
145     \ExplSyntaxOn
146     \cs_if_free:NT \pgfsyspdfmark
147     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
148     \ExplSyntaxOff
149   }
150   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
151 }

```

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.

```

152 \ProvideDocumentCommand \iddots { }
153 {
154   \mathinner
155   {
156     \tex_mkern:D 1 mu
157     \box_move_up:nn { 1 pt } { \hbox:n { . } }
158     \tex_mkern:D 2 mu
159     \box_move_up:nn { 4 pt } { \hbox:n { . } }
160     \tex_mkern:D 2 mu
161     \box_move_up:nn { 7 pt }

```

```

162         { \vbox:n { \kern 7 pt \hbox:n { . } } }
163     \tex_mkern:D 1 mu
164 }
165 }

```

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.

```

166 \hook_gput_code:nnn { begindocument } { . }
167 {
168     \IfPackageLoadedTF { booktabs }
169     { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
170     { }
171 }
172 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
173 {
174     \cs_set_eq:NN \@@_old_pgfulil@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`).

```

175     \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
176     {
177         \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
178         { \@@_old_pgfulil@check@rerun { ##1 } { ##2 } }
179     }
180 }

```

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

```

181 \hook_gput_code:nnn { begindocument } { . }
182 {
183     \IfPackageLoadedTF { colortbl }
184     { }
185     {

```

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.

```

186         \cs_set_protected:Npn \CT@arc@ { }
187         \cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
188         \cs_set:Npn \CT@arc #1 #2
189         {
190             \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
191             { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
192         }

```

Idem for `\CT@drs@`.

```

193         \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
194         \cs_set:Npn \CT@drs #1 #2
195         {
196             \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
197             { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
198         }
199         \cs_set:Npn \hline
200         {
201             \noalign { \ifnum 0 = ` } \fi
202             \cs_set_eq:NN \hskip \vskip
203             \cs_set_eq:NN \vrule \hrule
204             \cs_set_eq:NN \@width \@height
205             { \CT@arc@ \vline }
206             \futurelet \reserved@a
207             \@xhline
208         }

```

```

209     }
210 }

```

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.

```

211 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
212 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
213 {
214   \int_if_zero:nT \l_@@_first_col_int { \omit & }
215   \int_compare:nNnT { #1 } > 1 { \multispan { \int_eval:n { #1 - 1 } } & }
216   \multispan { \int_eval:n { #2 - #1 + 1 } }
217   {
218     \CT@arc@
219     \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

220     \skip_horizontal:N \c_zero_dim
221 }

```

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`.

```

222   \everycr { }
223   \cr
224   \noalign { \skip_vertical:N -\arrayrulewidth }
225 }

```

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.

```

226 \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`.

```

227 { \@@_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*.

```

228 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
229 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
230 {
231   \tl_if_empty:nTF { #3 }
232   { \@@_cline_iii:w #1|#2-#2 \q_stop }
233   { \@@_cline_ii:w #1|#2-#3 \q_stop }
234 }
235 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
236 { \@@_cline_iii:w #1|#2-#3 \q_stop }
237 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
238 {

```

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).

```

239   \int_compare:nNnT { #1 } < { #2 }
240   { \multispan { \int_eval:n { #2 - #1 } } & }
241   \multispan { \int_eval:n { #3 - #2 + 1 } }
242   {
243     \CT@arc@
244     \leaders \hrule \@height \arrayrulewidth \hfill
245     \skip_horizontal:N \c_zero_dim
246   }

```

¹See question 99041 on TeX StackExchange.

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

```

247 \peek_meaning_remove_ignore_spaces:NTF \cline
248 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
249 { \everycr { } \cr }
250 }
251 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following command is a small shortcut.

```

252 \cs_new:Npn \@@_math_toggle_token:
253 { \bool_if:NF \l_@@_tabular_bool \c_math_toggle_token }

```

```

254 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
255 {
256   \tl_if_blank:NF { #1 }
257   {
258     \tl_if_head_eq_meaning:nNTF { #1 } [
259       { \cs_set:Npn \CT@arc@ { \color #1 } }
260       { \cs_set:Npn \CT@arc@ { \color { #1 } } }
261     ]
262   }
263 \cs_generate_variant:Nn \@@_set_CT@arc@:n { V }

```

```

264 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
265 {
266   \tl_if_head_eq_meaning:nNTF { #1 } [
267     { \cs_set:Npn \CT@drsc@ { \color #1 } }
268     { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
269   ]
270 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { V }

```

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

```

271 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
272 {
273   \tl_if_head_eq_meaning:nNTF { #2 } [
274     { #1 #2 }
275     { #1 { #2 } }
276   ]
277 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N V }

```

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

```

278 \cs_new_protected:Npn \@@_color:n #1
279 { \tl_if_blank:NF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
280 \cs_generate_variant:Nn \@@_color:n { V }

```

```

281 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

```

282 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
283 {
284   \tl_set_rescan:Nno
285   #1
286   {
287     \char_set_catcode_other:N >
288     \char_set_catcode_other:N <
289   }
290   #1
291 }

```


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.

```

292 \cs_set_eq:NN \@_old_mkpream: \@mkpream
293 \cs_set_protected:Npn \@_mkpream: #1
294 {

```

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

```

295 \@_mkpream_colortbl:
296 \gdef\@preamble{} \lastchclass 4 \@firstamtrue
297 \let\@sharp\relax
298 \def\@startpbox##1{\unexpanded\expandafter{\expandafter
299 \@startpbox\expandafter{##1}}}\let\@endpbox\relax
300 \let\do@row@strut\relax
301 \let\ar@align@mcell\relax
302 \@temptokena{#1} % \@tempswatrue
303 % \@whilesw@if@tempswa\fi{\@tempswafalse\the\NC@list}%
304 \count@\m@ne
305 \let\the@toks\relax
306 \prepnext@tok

```

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

```

307 \exp_args:NV \tl_map_variable:NNn \@temptokena \@nextchar
308 {\@testpach
309 \ifcase \@chclass \@classz \or \@classi \or \@classii
310 \or \save@decl \or \or \@classv \or \@classvi
311 \or \@classvii \or \@classviii
312 \or \@classx
313 \or \@classx \fi
314 \@lastchclass\@chclass}%
315 \ifcase\@lastchclass
316 \@acol \or
317 \or
318 \@acol \or
319 \@preamerr \thr@@ \or
320 \@preamerr \tw@ \@addtopreamble\@sharp \or
321 \or
322 \else \@preamerr \@ne \fi
323 \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).

```

324 \cs_gset_eq:NN \@mkpream \@_old_mkpream:
325 }

```

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...

```

326 \hook_gput_code:nnn { begindocument } { . }
327 {
328 \bool_if:NTF \c_@@_revtex_bool
329 { \cs_new_protected:Npn \@_redefine_mkpream: { } }
330 {
331 \IfPackageLoadedTF { arydshln }
332 { \cs_new_protected:Npn \@_redefine_mkpream: { } }
333 {
334 \cs_new_protected:Npn \@_redefine_mkpream:
335 { \cs_set_eq:NN \@mkpream \@_mkpream: }
336 }
337 }
338 }

339 \cs_new_protected:Npn \@_mkpream_colortbl: { }
340 \hook_gput_code:nnn { begindocument } { . }

```

```

341 {
342   \IfPackageLoadedTF { colortbl }
343   {
344     \cs_set_protected:Npn \@@_mkpream_colortbl:
345     {

```

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.

```

346       \cs_set_eq:NN \CT@setup \relax
347       \cs_set_eq:NN \CT@color \relax
348       \cs_set_eq:NN \CT@do@color \relax
349       \cs_set_eq:NN \color \relax
350       \cs_set_eq:NN \CT@column@color \relax
351       \cs_set_eq:NN \CT@row@color \relax
352       \cs_set_eq:NN \CT@cell@color \relax
353     }
354   }
355   { }
356 }

```

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.

```

357 \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).

```

358 \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.

```

359 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
360 { \int_use:N \g_@@_env_int }

```

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

```

361 \cs_new_protected:Npn \@@_qpoint:n #1
362 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

363 \bool_new:N \l_@@_tabular_bool

```

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

```

364 \bool_new:N \g_@@_delims_bool
365 \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.

```
366 \bool_new:N \l_@@_preamble_bool
367 \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.

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

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

```
369 \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.

```
370 \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).

```
371 \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.).

```
372 \dim_new:N \l_@@_col_width_dim
373 \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.

```
374 \int_new:N \g_@@_row_total_int
375 \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).

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

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

```
377 \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.

```
378 \str_new:N \l_@@_hpos_cell_str
379 \str_set:Nn \l_@@_hpos_cell_str { c }
```

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

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

Idem for the mono-row blocks.

```
381 \dim_new:N \g_@@_blocks_ht_dim
382 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

```
384 \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`.

```
385 \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.

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

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

```
387 \bool_new:N \l_@@_notes_detect_duplicates_bool
388 \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.

```
389 \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).

```
390 \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).

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

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

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

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

```
393 \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.

```
394 \bool_new:N \l_@@_X_bool
395 \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_}`).

```
396 \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.

```
397 \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.

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

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

```
400 \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}`).

```
401 \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`).

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

For `\multicolumn`.

```
403 \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`.

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

```
405 \tl_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 `:`.

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

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

```
408 \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`).

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

```
410 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
411 {
```

```
412   \if_mode_math: \else:
```

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

```
414   \fi:
```

```
415 }
```

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.

```
416 \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”.

```
417 \colorlet { nicematrix-last-col } { . }
```

```
418 \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`).

```
419 \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*.

```
420 \tl_new:N \g_@@_com_or_env_str
421 \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).

```
422 \cs_new:Npn \@_full_name_env:
423 {
424   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
425   { command \space \c_backslash_str \g_@@_name_env_str }
426   { environment \space \{ \g_@@_name_env_str \} }
427 }
```

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

```
428 \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.

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

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

```
430
```

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

```
431 \tl_new:N \g_@@_pre_code_before_tl
432 \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.

```
433 \tl_new:N \g_@@_pre_code_after_tl
434 \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.

```
435 \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.

```
436 \int_new:N \l_@@_old_iRow_int
437 \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).

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

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

```
439 \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.

```
440 \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.

```
441 \bool_new:N \l_@@_X_columns_aux_bool
442 \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.

```
443 \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).

```
444 \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).

```
445 \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`).

```
446 \tl_new:N \l_@@_code_before_tl
447 \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).

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

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

```
449 \dim_new:N \l_@@_x_initial_dim
450 \dim_new:N \l_@@_y_initial_dim
451 \dim_new:N \l_@@_x_final_dim
452 \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.

```
453 \dim_new:N \l_@@_tmpc_dim
454 \dim_new:N \l_@@_tmpd_dim
```

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

```
455 \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”.

```
456 \dim_new:N \g_@@_width_last_col_dim
457 \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.

```
458 \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!).

```
459 \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}`.

```
460 \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.

```
461 \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.

```
462 \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`).

```
463 \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.

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

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

```
465 \seq_new:N \g_@@_multicolumn_cells_seq
466 \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`).

```
467 \int_new:N \l_@@_row_min_int
468 \int_new:N \l_@@_row_max_int
469 \int_new:N \l_@@_col_min_int
470 \int_new:N \l_@@_col_max_int
```

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

```
471 \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).

```
472 \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`.

```
473 \tl_new:N \l_@@_fill_tl
474 \tl_new:N \l_@@_opacity_tl
475 \tl_new:N \l_@@_draw_tl
476 \seq_new:N \l_@@_tikz_seq
477 \clist_new:N \l_@@_borders_clist
478 \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.

```
479 \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`.

```
480 \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.

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

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

```
482 \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`).

```
483 \str_new:N \l_@@_hpos_block_str
484 \str_set:Nn \l_@@_hpos_block_str { c }
485 \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.

```
486 \bool_new:N \@@_nocolor_used_bool
```

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

```
487 \str_new:N \l_@@_vpos_of_block_str
488 \str_set:Nn \l_@@_vpos_of_block_str { c }
```

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

```
489 \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).

```
490 \bool_new:N \l_@@_vlines_block_bool
491 \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.

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

493 \dim_new:N \l_@@_submatrix_extra_height_dim
494 \dim_new:N \l_@@_submatrix_left_xshift_dim
495 \dim_new:N \l_@@_submatrix_right_xshift_dim
496 \clist_new:N \l_@@_hlines_clist
497 \clist_new:N \l_@@_vlines_clist
498 \clist_new:N \l_@@_submatrix_hlines_clist
499 \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}`).

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

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

```
501 \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`).

```
502 \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.

```
503 \int_new:N \l_@@_first_row_int
504 \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.

```
505 \int_new:N \l_@@_first_col_int
506 \int_set:Nn \l_@@_first_col_int 1
```

- **Last row**

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

```
507 \int_new:N \l_@@_last_row_int
508 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
510 \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`.

```
511 \int_new:N \l_@@_last_col_int
512 \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.

```
513 \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`).

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

Some utilities

```
515 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
516 {
517   \tl_set:Nn \l_tmpa_tl { #1 }
518   \tl_set:Nn \l_tmpb_tl { #2 }
519 }
```

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.

```

520 \cs_new_protected:Npn \@@_expand_clist:N #1
521 {
522   \clist_if_in:NnF #1 { all }
523   {
524     \clist_clear:N \l_tmpa_clist
525     \clist_map_inline:Nn #1
526     {
527       \tl_if_in:nnTF { ##1 } { - }
528       { \@@_cut_on_hyphen:w ##1 \q_stop }
529       {
530         \tl_set:Nn \l_tmpa_tl { ##1 }
531         \tl_set:Nn \l_tmpb_tl { ##1 }
532       }
533       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
534       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
535     }
536     \tl_set_eq:NN #1 \l_tmpa_clist
537   }
538 }

```

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.

```

539 \hook_gput_code:nnn { begindocument } { . }
540 {
541   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
542   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
543   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
544 }

```

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.

```

545 \newcounter { tabulernote }
546 \seq_new:N \g_@@_notes_seq
547 \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.

```

548 \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.

```

549 \seq_new:N \l_@@_notes_labels_seq
550 \newcounter{nicematrix_draft}
551 \cs_new_protected:Npn \@@_notes_format:n #1
552 {
553   \setcounter { nicematrix_draft } { #1 }
554   \@@_notes_style:n { nicematrix_draft }
555 }

```

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

```

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

```

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

```

557 \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`.

```

558 \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`.

```

559 \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`).

```

560 \hook_gput_code:nnn { begindocument } { . }
561 {
562   \IfPackageLoadedTF { enumitem }
563   {

```

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.

```

564     \newlist { tabularnotes } { enumerate } { 1 }
565     \setlist [ tabularnotes ]
566     {
567       topsep = Opt ,
568       noitemsep ,
569       leftmargin = * ,
570       align = left ,
571       labelsep = Opt ,
572       label =
573       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
574     }
575     \newlist { tabularnotes* } { enumerate* } { 1 }
576     \setlist [ tabularnotes* ]
577     {
578       afterlabel = \nobreak ,
579       itemjoin = \quad ,
580       label =
581       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
582     }

```

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`.

```

583     \NewDocumentCommand \tabularnote { o m }
584     {
585       \bool_if:nT { \cs_if_exist_p:N \@capytype || \l_@@_in_env_bool }
586       {
587         \bool_if:nTF { ! \l_@@_tabular_bool && \l_@@_in_env_bool }
588         { \@@_error:n { tabularnote~forbidden } }
589         {
590           \bool_if:NTF \l_@@_in_caption_bool
591           { \@@_tabularnote_caption:nn { #1 } { #2 } }
592           { \@@_tabularnote:nn { #1 } { #2 } }
593         }
594       }
595     }
596   }
597   {
598     \NewDocumentCommand \tabularnote { o m }
599     {
600       \@@_error_or_warning:n { enumitem~not~loaded }
601       \@@_gredirect_none:n { enumitem~not~loaded }
602     }
603   }
604 }

```

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`.

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

644     \hbox_set:Nn \l_tmpa_box
645     {

```

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

```

646         \@@_notes_label_in_tabular:n
647         {

```

```

648         \seq_use:Nnnn
649         \l_@@_notes_labels_seq { , } { , } { , }
650     }
651 }

```

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

```

652     \int_gsub:Nn \c@tabularnote { 1 }
653     \int_set_eq:NN \l_tmpa_int \c@tabularnote
654     \refstepcounter { tabularnote }
655     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
656     { \int_gincr:N \c@tabularnote }
657     \seq_clear:N \l_@@_notes_labels_seq
658     \bool_lazy_or:nnTF
659     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { c } }
660     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { r } }
661     {
662         \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).

```

663         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
664     }
665     { \box_use:N \l_tmpa_box }
666 }
667 }

```

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`.

```

668 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
669 {
670     \bool_if:NTF \g_@@_caption_finished_bool
671     {
672         \int_compare:nNnT
673         \c@tabularnote = \g_@@_notes_caption_int
674         { \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`!

```

675         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } } { { #2 } }
676         { \@@_error:n { Identical~notes~in~caption } }
677     }
678 }

```

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

```

679     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } } { { #2 } }
680     {

```

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.

```

681         \bool_gset_true:N \g_@@_caption_finished_bool
682         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
683         \int_gzero:N \c@tabularnote
684     }
685     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } } { { #2 } } }
686 }

```


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!

```

687 \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
688 \seq_put_right:Nx \l_@@_notes_labels_seq
689 {
690   \tl_if_novalue:nTF { #1 }
691     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
692     { #1 }
693 }
694 \peek_meaning:NF \tabularnote
695 {
696   \@@_notes_label_in_tabular:n
697     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
698   \seq_clear:N \l_@@_notes_labels_seq
699 }
700 }

701 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
702 { \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.

```

703 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
704 {
705   \begin { pgfscope }
706   \pgfset
707   {
708     inner~sep = \c_zero_dim ,
709     minimum~size = \c_zero_dim
710   }
711   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
712   \pgfnode
713   { rectangle }
714   { center }
715   {
716     \vbox_to_ht:nn
717       { \dim_abs:n { #5 - #3 } }
718       {
719         \vfill
720         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
721       }
722   }
723   { #1 }
724   { }
725   \end { pgfscope }
726 }

```

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.

```

727 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
728 {
729   \begin { pgfscope }
730   \pgfset
731   {
732     inner~sep = \c_zero_dim ,
733     minimum~size = \c_zero_dim

```

```

734     }
735     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
736     \pgfpointdiff { #3 } { #2 }
737     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
738     \pgfnode
739     { rectangle }
740     { center }
741     {
742         \vbox_to_ht:nn
743         { \dim_abs:n \l_tmpb_dim }
744         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
745     }
746     { #1 }
747     { }
748     \end { pgfscope }
749 }

```

8 The options

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

```

750 \tl_new:N \l_@@_caption_tl
751 \tl_new:N \l_@@_short_caption_tl
752 \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).

```

753 \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.

```

754 \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`.

```

755 \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`).

```

756 \dim_new:N \l_@@_cell_space_top_limit_dim
757 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

758 \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.

```

759 \dim_new:N \l_@@_xdots_inter_dim
760 \hook_gput_code:nnn { begindocument } { . }
761 { \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).

```

762 \dim_new:N \l_@@_xdots_shorten_start_dim
763 \dim_new:N \l_@@_xdots_shorten_end_dim
764 \hook_gput_code:nnn { begindocument } { . }
765 {
766   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
767   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
768 }

```

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.

```

769 \dim_new:N \l_@@_xdots_radius_dim
770 \hook_gput_code:nnn { begindocument } { . }
771 { \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.

```

772 \tl_new:N \l_@@_xdots_line_style_tl
773 \tl_const:Nn \c_@@_standard_tl { standard }
774 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

```

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

```

775 \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).

```

776 \tl_new:N \l_@@_baseline_tl
777 \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`).

```

778 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

779 \bool_new:N \l_@@_parallelize_diags_bool
780 \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.

```

781 \clist_new:N \l_@@_corners_clist

```

```

782 \dim_new:N \l_@@_notes_above_space_dim
783 \hook_gput_code:nnn { begindocument } { . }
784 { \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.).

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

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

```
786 \bool_new:N \l_@@_respect_arraystretch_bool
```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

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

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

```
788 \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.

```
789 \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”.

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

```
791 \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).

```
792 \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`).

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

```
794 \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`.

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

```
796 \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.

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

```
798 \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 “:”.

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

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

```
800 \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.

```

801 \bool_new:N \l_@@_delimiters_max_width_bool

802 \keys_define:nn { NiceMatrix / xdots }
803 {
804   shorten-start .code:n =
805     \hook_gput_code:nnn { begindocument } { . }
806     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
807   shorten-end .code:n =
808     \hook_gput_code:nnn { begindocument } { . }
809     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
810   shorten-start .value_required:n = true ,
811   shorten-end .value_required:n = true ,
812   shorten .code:n =
813     \hook_gput_code:nnn { begindocument } { . }
814     {
815       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
816       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
817     } ,
818   shorten .value_required:n = true ,
819   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
820   horizontal-labels .default:n = true ,
821   line-style .code:n =
822     {
823       \bool_lazy_or:nnTF
824         { \cs_if_exist_p:N \tikzpicture }
825         { \str_if_eq_p:nn { #1 } { standard } }
826         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
827         { \@@_error:n { bad-option-for-line-style } }
828     } ,
829   line-style .value_required:n = true ,
830   color .tl_set:N = \l_@@_xdots_color_tl ,
831   color .value_required:n = true ,
832   radius .code:n =
833     \hook_gput_code:nnn { begindocument } { . }
834     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
835   radius .value_required:n = true ,
836   inter .code:n =
837     \hook_gput_code:nnn { begindocument } { . }
838     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
839   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}`.

```

840   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
841   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
842   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).

```

843   draw-first .code:n = \prg_do_nothing: ,
844   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
845 }

```

```

846 \keys_define:nn { NiceMatrix / rules }
847 {
848   color .tl_set:N = \l_@@_rules_color_tl ,
849   color .value_required:n = true ,
850   width .dim_set:N = \arrayrulewidth ,
851   width .value_required:n = true ,
852   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
853 }

```

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

```

854 \keys_define:nn { NiceMatrix / Global }
855 {
856   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
857   rounded-corners .default:n = 4 pt ,
858   custom-line .code:n = \@@_custom_line:n { #1 } ,
859   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
860   rules .value_required:n = true ,
861   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
862   standard-cline .default:n = true ,
863   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
864   cell-space-top-limit .value_required:n = true ,
865   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
866   cell-space-bottom-limit .value_required:n = true ,
867   cell-space-limits .meta:n =
868   {
869     cell-space-top-limit = #1 ,
870     cell-space-bottom-limit = #1 ,
871   } ,
872   cell-space-limits .value_required:n = true ,
873   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
874   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
875   light-syntax .default:n = true ,
876   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
877   end-of-row .value_required:n = true ,
878   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
879   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
880   last-row .int_set:N = \l_@@_last_row_int ,
881   last-row .default:n = -1 ,
882   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
883   code-for-first-col .value_required:n = true ,
884   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
885   code-for-last-col .value_required:n = true ,
886   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
887   code-for-first-row .value_required:n = true ,
888   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
889   code-for-last-row .value_required:n = true ,
890   hlines .clist_set:N = \l_@@_hlines_clist ,
891   vlimes .clist_set:N = \l_@@_vlimes_clist ,
892   hlines .default:n = all ,
893   vlimes .default:n = all ,
894   vlimes-in-sub-matrix .code:n =
895   {
896     \tl_if_single_token:nTF { #1 }
897     {
898       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
899       { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

900     { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
901   }
902   { \@@_error:n { One~letter~allowed } }
903 } ,

```

```

904     vlines-in-sub-matrix .value_required:n = true ,
905     hvlines .code:n =
906     {
907         \bool_set_true:N \l_@@_hvlines_bool
908         \clist_set:Nn \l_@@_vlines_clist { all }
909         \clist_set:Nn \l_@@_hlines_clist { all }
910     } ,
911     hvlines-except-borders .code:n =
912     {
913         \clist_set:Nn \l_@@_vlines_clist { all }
914         \clist_set:Nn \l_@@_hlines_clist { all }
915         \bool_set_true:N \l_@@_hvlines_bool
916         \bool_set_true:N \l_@@_except_borders_bool
917     } ,
918     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.

```

919     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
920     renew-dots .value_forbidden:n = true ,
921     nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
922     create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
923     create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
924     create-extra-nodes .meta:n =
925     { create-medium-nodes , create-large-nodes } ,
926     left-margin .dim_set:N = \l_@@_left_margin_dim ,
927     left-margin .default:n = \arraycolsep ,
928     right-margin .dim_set:N = \l_@@_right_margin_dim ,
929     right-margin .default:n = \arraycolsep ,
930     margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
931     margin .default:n = \arraycolsep ,
932     extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
933     extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
934     extra-margin .meta:n =
935     { extra-left-margin = #1 , extra-right-margin = #1 } ,
936     extra-margin .value_required:n = true ,
937     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
938     respect-arraystretch .default:n = true ,
939     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
940     pgf-node-code .value_required:n = true
941 }

```

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

```

942 \keys_define:nn { NiceMatrix / Env }
943 {
944     corners .clist_set:N = \l_@@_corners_clist ,
945     corners .default:n = { NW , SW , NE , SE } ,
946     code-before .code:n =
947     {
948         \tl_if_empty:nF { #1 }
949         {
950             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
951             \bool_set_true:N \l_@@_code_before_bool
952         }
953     } ,
954     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}`.

```

955     c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,

```

```

956 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
957 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
958 baseline .tl_set:N = \l_@@_baseline_tl ,
959 baseline .value_required:n = true ,
960 columns-width .code:n =
961   \tl_if_eq:nnTF { #1 } { auto }
962     { \bool_set_true:N \l_@@_auto_columns_width_bool }
963     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
964 columns-width .value_required:n = true ,
965 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.

```

966   \legacy_if:nF { measuring@ }
967   {
968     \str_set:Nx \l_tmpa_str { #1 }
969     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
970       { \@@_error:nn { Duplicate~name } { #1 } }
971       { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
972     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
973   } ,
974 name .value_required:n = true ,
975 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
976 code-after .value_required:n = true ,
977 color-inside .code:n =
978   \bool_set_true:N \l_@@_color_inside_bool
979   \bool_set_true:N \l_@@_code_before_bool ,
980 color-inside .value_forbidden:n = true ,
981 colortbl-like .meta:n = color-inside
982 }
983 \keys_define:nn { NiceMatrix / notes }
984 {
985   para .bool_set:N = \l_@@_notes_para_bool ,
986   para .default:n = true ,
987   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
988   code-before .value_required:n = true ,
989   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
990   code-after .value_required:n = true ,
991   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
992   bottomrule .default:n = true ,
993   style .cs_set:Np = \@@_notes_style:n #1 ,
994   style .value_required:n = true ,
995   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
996   label-in-tabular .value_required:n = true ,
997   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
998   label-in-list .value_required:n = true ,
999   enumitem-keys .code:n =
1000   {
1001     \hook_gput_code:nnn { begindocument } { . }
1002     {
1003       \IfPackageLoadedTF { enumitem }
1004         { \setlist* [ tabularnotes ] { #1 } }
1005         { }
1006     }
1007   } ,
1008   enumitem-keys .value_required:n = true ,
1009   enumitem-keys-para .code:n =
1010   {
1011     \hook_gput_code:nnn { begindocument } { . }
1012     {
1013       \IfPackageLoadedTF { enumitem }
1014         { \setlist* [ tabularnotes* ] { #1 } }
1015         { }
1016     }

```



```

1017     } ,
1018     enumitem-keys-para .value_required:n = true ,
1019     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1020     detect-duplicates .default:n = true ,
1021     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1022 }
1023 \keys_define:nn { NiceMatrix / delimiters }
1024 {
1025     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1026     max-width .default:n = true ,
1027     color .tl_set:N = \l_@@_delimiters_color_tl ,
1028     color .value_required:n = true ,
1029 }

```

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

```

1030 \keys_define:nn { NiceMatrix }
1031 {
1032     NiceMatrixOptions .inherit:n =
1033     { NiceMatrix / Global } ,
1034     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1035     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1036     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1037     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1038     SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1039     CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1040     CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1041     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1042     NiceMatrix .inherit:n =
1043     {
1044         NiceMatrix / Global ,
1045         NiceMatrix / Env ,
1046     } ,
1047     NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1048     NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1049     NiceTabular .inherit:n =
1050     {
1051         NiceMatrix / Global ,
1052         NiceMatrix / Env
1053     } ,
1054     NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1055     NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1056     NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1057     NiceArray .inherit:n =
1058     {
1059         NiceMatrix / Global ,
1060         NiceMatrix / Env ,
1061     } ,
1062     NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1063     NiceArray / rules .inherit:n = NiceMatrix / rules ,
1064     pNiceArray .inherit:n =
1065     {
1066         NiceMatrix / Global ,
1067         NiceMatrix / Env ,
1068     } ,
1069     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1070     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1071 }

```

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

```

1072 \keys_define:nn { NiceMatrix / NiceMatrixOptions }

```

```

1073 {
1074   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1075   delimiters / color .value_required:n = true ,
1076   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1077   delimiters / max-width .default:n = true ,
1078   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1079   delimiters .value_required:n = true ,
1080   width .dim_set:N = \l_@@_width_dim ,
1081   width .value_required:n = true ,
1082   last-col .code:n =
1083     \tl_if_empty:nF { #1 }
1084     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1085     \int_zero:N \l_@@_last_col_int ,
1086   small .bool_set:N = \l_@@_small_bool ,
1087   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.

```

1088   renew-matrix .code:n = \@@_renew_matrix: ,
1089   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}`.

```

1090   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.

```

1091   columns-width .code:n =
1092     \tl_if_eq:nnTF { #1 } { auto }
1093     { \@@_error:n { Option-auto-for-columns-width } }
1094     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1095   allow-duplicate-names .code:n =
1096     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1097   allow-duplicate-names .value_forbidden:n = true ,
1098   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1099   notes .value_required:n = true ,
1100   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1101   sub-matrix .value_required:n = true ,
1102   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1103   matrix / columns-type .value_required:n = true ,
1104   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1105   caption-above .default:n = true ,
1106   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1107 }

```

`\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.

```

1108 \NewDocumentCommand \NiceMatrixOptions { m }
1109 { \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.

```

1110 \keys_define:nn { NiceMatrix / NiceMatrix }
1111 {
1112   last-col .code:n = \tl_if_empty:nTF { #1 }
1113     {

```

```

1114         \bool_set_true:N \l_@@_last_col_without_value_bool
1115         \int_set:Nn \l_@@_last_col_int { -1 }
1116     }
1117     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1118     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1119     columns-type .value_required:n = true ,
1120     l .meta:n = { columns-type = l } ,
1121     r .meta:n = { columns-type = r } ,
1122     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1123     delimiters / color .value_required:n = true ,
1124     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1125     delimiters / max-width .default:n = true ,
1126     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1127     delimiters .value_required:n = true ,
1128     small .bool_set:N = \l_@@_small_bool ,
1129     small .value_forbidden:n = true ,
1130     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1131 }

```

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

```

1132 \keys_define:nn { NiceMatrix / NiceArray }
1133 {

```

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.

```

1134     small .bool_set:N = \l_@@_small_bool ,
1135     small .value_forbidden:n = true ,
1136     last-col .code:n = \tl_if_empty:nF { #1 }
1137         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1138         \int_zero:N \l_@@_last_col_int ,
1139     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1140     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1141     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1142 }
1143 \keys_define:nn { NiceMatrix / pNiceArray }
1144 {
1145     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1146     last-col .code:n = \tl_if_empty:nF { #1 }
1147         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1148         \int_zero:N \l_@@_last_col_int ,
1149     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1150     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1151     delimiters / color .value_required:n = true ,
1152     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1153     delimiters / max-width .default:n = true ,
1154     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1155     delimiters .value_required:n = true ,
1156     small .bool_set:N = \l_@@_small_bool ,
1157     small .value_forbidden:n = true ,
1158     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1159     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1160     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1161 }

```

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

```

1162 \keys_define:nn { NiceMatrix / NiceTabular }
1163 {

```

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.

```

1164     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1165           \bool_set_true:N \l_@@_width_used_bool ,
1166     width .value_required:n = true ,
1167     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1168     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1169     tabularnote .value_required:n = true ,
1170     caption .tl_set:N = \l_@@_caption_tl ,
1171     caption .value_required:n = true ,
1172     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1173     short-caption .value_required:n = true ,
1174     label .tl_set:N = \l_@@_label_tl ,
1175     label .value_required:n = true ,
1176     last-col .code:n = \tl_if_empty:nF {#1}
1177           { \@@_error:n { last-col-non-empty-for-NiceArray } }
1178           \int_zero:N \l_@@_last_col_int ,
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~NiceTabular }
1182   }

```

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`

```

1183 \keys_define:nn { NiceMatrix / CodeAfter }
1184 {
1185     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1186     delimiters / color .value_required:n = true ,
1187     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1188     rules .value_required:n = true ,
1189     xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1190     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1191     sub-matrix .value_required:n = true ,
1192     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1193 }

```

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}`).

```

1194 \cs_new_protected:Npn \@@_cell_begin:w
1195 {

```

`\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.

```

1196     \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).

```

1197     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

1198     \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.

```
1199 \int_compare:nNnT \c@jCol = 1
1200 { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the `\@@_cell_end:` (and the potential `\c_math_toggle_token` also).

```
1201 \hbox_set:Nw \l_@@_cell_box
1202 \bool_if:NF \l_@@_tabular_bool
1203 {
1204   \c_math_toggle_token
1205   \bool_if:NT \l_@@_small_bool \scriptstyle
1206 }
1207 \g_@@_row_style_tl
```

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).

The codes `\l_@@_code_for_first_row_tl` and `al` don't apply in the corners of the matrix.

```
1208 \int_if_zero:nTF \c@iRow
1209 {
1210   \int_compare:nNnT \c@jCol > 0
1211   {
1212     \l_@@_code_for_first_row_tl
1213     \xglobal \colorlet { nicematrix-first-row } { . }
1214   }
1215 }
1216 {
1217   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1218   {
1219     \l_@@_code_for_last_row_tl
1220     \xglobal \colorlet { nicematrix-last-row } { . }
1221   }
1222 }
1223 }
```

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.

```
1224 \cs_new_protected:Npn \@@_begin_of_row:
1225 {
1226   \int_gincr:N \c@iRow
1227   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1228   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1229   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1230   \pgfpicture
1231   \pgfrememberpicturepositiononpagetrue
1232   \pgfcoordinate
1233   { \@@_env: - row - \int_use:N \c@iRow - base }
1234   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1235   \str_if_empty:NF \l_@@_name_str
1236   {
1237     \pgfnodelalias
1238     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1239     { \@@_env: - row - \int_use:N \c@iRow - base }
1240   }
1241   \endpgfpicture
1242 }
```

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.

```

1243 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1244 {
1245   \int_if_zero:nTF \c@iRow
1246   {
1247     \dim_gset:Nn \g_@@_dp_row_zero_dim
1248     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1249     \dim_gset:Nn \g_@@_ht_row_zero_dim
1250     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1251   }
1252   {
1253     \int_compare:nNnT \c@iRow = 1
1254     {
1255       \dim_gset:Nn \g_@@_ht_row_one_dim
1256       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1257     }
1258   }
1259 }

1260 \cs_new_protected:Npn \@@_rotate_cell_box:
1261 {
1262   \box_rotate:Nn \l_@@_cell_box { 90 }
1263   \bool_if:NTF \g_@@_rotate_c_bool
1264   {
1265     \hbox_set:Nn \l_@@_cell_box
1266     {
1267       \c_math_toggle_token
1268       \vcenter { \box_use:N \l_@@_cell_box }
1269       \c_math_toggle_token
1270     }
1271   }
1272   {
1273     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1274     {
1275       \vbox_set_top:Nn \l_@@_cell_box
1276       {
1277         \vbox_to_zero:n { }
1278         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1279         \box_use:N \l_@@_cell_box
1280       }
1281     }
1282   }
1283   \bool_gset_false:N \g_@@_rotate_bool
1284   \bool_gset_false:N \g_@@_rotate_c_bool
1285 }

1286 \cs_new_protected:Npn \@@_adjust_size_box:
1287 {
1288   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1289   {
1290     \box_set_wd:Nn \l_@@_cell_box
1291     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1292     \dim_gzero:N \g_@@_blocks_wd_dim
1293   }
1294   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1295   {
1296     \box_set_dp:Nn \l_@@_cell_box
1297     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1298     \dim_gzero:N \g_@@_blocks_dp_dim
1299   }
1300   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1301   {

```

```

1302     \box_set_ht:Nn \l_@@_cell_box
1303     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1304     \dim_gzero:N \g_@@_blocks_ht_dim
1305   }
1306 }
1307 \cs_new_protected:Npn \@@_cell_end:
1308 {
1309   \@@_math_toggle_token:
1310   \hbox_set_end:
1311   \@@_cell_end_i:
1312 }
1313 \cs_new_protected:Npn \@@_cell_end_i:
1314 {

```

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.

```

1315   \g_@@_cell_after_hook_tl
1316   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1317   \@@_adjust_size_box:
1318   \box_set_ht:Nn \l_@@_cell_box
1319   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1320   \box_set_dp:Nn \l_@@_cell_box
1321   { \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”).

```

1322   \@@_update_max_cell_width:

```

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

```

1323   \@@_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.

```

1324   \bool_if:NTF \g_@@_empty_cell_bool
1325   { \box_use_drop:N \l_@@_cell_box }
1326   {
1327     \bool_lazy_or:nnTF
1328     \g_@@_not_empty_cell_bool
1329     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1330     \@@_node_for_cell:
1331     { \box_use_drop:N \l_@@_cell_box }
1332   }
1333   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }

```

```

1334 \bool_gset_false:N \g_@@_empty_cell_bool
1335 \bool_gset_false:N \g_@@_not_empty_cell_bool
1336 }

```

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

```

1337 \cs_new_protected:Npn \@@_update_max_cell_width:
1338 {
1339   \dim_gset:Nn \g_@@_max_cell_width_dim
1340   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1341 }

```

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`).

```

1342 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1343 {
1344   \@@_math_toggle_token:
1345   \hbox_set_end:
1346   \bool_if:NF \g_@@_rotate_bool
1347   {
1348     \hbox_set:Nn \l_@@_cell_box
1349     {
1350       \makebox [ \l_@@_col_width_dim ] [ s ]
1351       { \hbox_unpack_drop:N \l_@@_cell_box }
1352     }
1353   }
1354   \@@_cell_end_i:
1355 }

```

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

```

1356 \pgfset
1357 {
1358   nicematrix / cell-node /.style =
1359   {
1360     inner~sep = \c_zero_dim ,
1361     minimum~width = \c_zero_dim
1362   }
1363 }
1364 \cs_new_protected:Npn \@@_node_for_cell:
1365 {
1366   \pgfpicture
1367   \pgfsetbaseline \c_zero_dim
1368   \pgfrememberpicturerepositiononpagetrue
1369   \pgfset { nicematrix / cell-node }
1370   \pgfnode
1371   { rectangle }
1372   { base }
1373   {

```

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

```

1374 \set@color
1375 \box_use_drop:N \l_@@_cell_box
1376 }
1377 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1378 { \l_@@_pgf_node_code_tl }
1379 \str_if_empty:NF \l_@@_name_str
1380 {
1381   \pgfnodealias
1382   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1383   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1384 }

```



```

1385 \endpgfpicture
1386 }

```

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.

```

1387 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1388 {
1389   \cs_new_protected:Npn \@@_patch_node_for_cell:
1390   {
1391     \hbox_set:Nn \l_@@_cell_box
1392     {
1393       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1394       \hbox_overlap_left:n
1395       {
1396         \pgfsys@markposition
1397         { \@@_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.

```

1398         #1
1399       }
1400       \box_use:N \l_@@_cell_box
1401       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1402       \hbox_overlap_left:n
1403       {
1404         \pgfsys@markposition
1405         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1406         #1
1407       }
1408     }
1409   }
1410 }

```

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

```

1411 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1412 {
1413   \@@_patch_node_for_cell:n
1414   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1415 }
1416 { \@@_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).

```

1417 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1418 {
1419   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1420   { g_@@_ #2 _ lines _ tl }
1421   {
1422     \use:c { @@ _ draw _ #2 : nnn }
1423     { \int_use:N \c@iRow }
1424     { \int_use:N \c@jCol }
1425     { \exp_not:n { #3 } }
1426   }
1427 }

```

```

1428 \cs_new_protected:Npn \@@_array:
1429 {

```

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).

```

1430   \@@_redefine_mkpream:
1431   \dim_set:Nn \col@sep
1432   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1433   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1434   { \cs_set_nopar:Npn \@halignto { } }
1435   { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1436   \@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.

```

1437   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1438 }

```

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.

```

1439 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1440 \cs_new_protected:Npn \@@_create_row_node:
1441 {
1442   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1443   {
1444     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1445     \@@_create_row_node_i:
1446   }
1447 }
1448 \cs_new_protected:Npn \@@_create_row_node_i:
1449 {

```

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

```

1450   \hbox
1451   {
1452     \bool_if:NT \l_@@_code_before_bool
1453     {
1454       \vtop
1455       {
1456         \skip_vertical:N 0.5\arrayrulewidth
1457         \pgfsys@markposition
1458         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1459         \skip_vertical:N -0.5\arrayrulewidth
1460       }
1461     }

```

```

1462 \pgfpicture
1463 \pgfrememberpicturepositiononpagetrue
1464 \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1465 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1466 \str_if_empty:NF \l_@@_name_str
1467 {
1468 \pgfnodealias
1469 { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1470 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1471 }
1472 \endpgfpicture
1473 }
1474 }

```

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

```

1475 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1476 \cs_new_protected:Npn \@@_everycr_i:
1477 {
1478 \int_gzero:N \c@jCol
1479 \bool_gset_false:N \g_@@_after_col_zero_bool
1480 \bool_if:NF \g_@@_row_of_col_done_bool
1481 {
1482 \@@_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).

```

1483 \tl_if_empty:NF \l_@@_hlines_clist
1484 {
1485 \tl_if_eq:NnF \l_@@_hlines_clist { all }
1486 {
1487 \exp_args:NNe
1488 \clist_if_in:NnT
1489 \l_@@_hlines_clist
1490 { \int_eval:n { \c@iRow + 1 } }
1491 }
1492 {

```

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.

```

1493 \int_compare:nNnT \c@iRow > { -1 }
1494 {
1495 \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@`.

```

1496 { \hrule height \arrayrulewidth width \c_zero_dim }
1497 }
1498 }
1499 }
1500 }
1501 }

```

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

```

1502 \cs_set_protected:Npn \@@_renew_dots:
1503 {
1504 \cs_set_eq:NN \ldots \@@_Ldots
1505 \cs_set_eq:NN \cdots \@@_Cdots
1506 \cs_set_eq:NN \vdots \@@_Vdots
1507 \cs_set_eq:NN \ddots \@@_Ddots
1508 \cs_set_eq:NN \iddots \@@_Iddots
1509 \cs_set_eq:NN \dots \@@_Ldots
1510 \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1511 }

```

```

1512 \cs_new_protected:Npn \@@_test_color_inside:
1513 {
1514   \bool_if:NF \l_@@_color_inside_bool
1515   {

```

We will issue an error only during the first run.

```

1516     \bool_if:NF \g_@@_aux_found_bool
1517     { \@@_error:n { without~color~inside } }
1518   }
1519 }

```

```

1520 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1521 \hook_gput_code:nnn { begindocument } { . }
1522 {
1523   \IfPackageLoadedTF { colortbl }
1524   {
1525     \cs_set_protected:Npn \@@_redefine_everycr:
1526     {
1527       \CT@everycr
1528       {
1529         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1530         \@@_everycr:
1531       }
1532     }
1533   }
1534   { }
1535 }

```

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.

```

1536 \hook_gput_code:nnn { begindocument } { . }
1537 {
1538   \IfPackageLoadedTF { booktabs }
1539   {
1540     \cs_new_protected:Npn \@@_patch_booktabs:
1541     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1542   }
1543   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1544 }

```

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

```

1545 \cs_new_protected:Npn \@@_pre_array_ii:
1546 {

```

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

```

1547   \int_gzero:N \g_@@_total_X_weight_int
1548   \@@_expand_clist:N \l_@@_hlines_clist
1549   \@@_expand_clist:N \l_@@_vlines_clist
1550   \@@_patch_booktabs:
1551   \box_clear_new:N \l_@@_cell_box
1552   \normalbaselines

```

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

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

```

1553     \bool_if:NT \l_@@_small_bool
1554     {
1555         \cs_set_nopar:Npn \arraystretch { 0.47 }
1556         \dim_set:Nn \arraycolsep { 1.45 pt }
1557     }

1558     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1559     {
1560         \tl_put_right:Nn \@@_begin_of_row:
1561         {
1562             \pgfsys@markposition
1563             { \@@_env: - row - \int_use:N \c@iRow - base }
1564         }
1565     }

```

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`.

```

1566     \cs_set_nopar:Npn \ialign
1567     {
1568         \@@_redefine_everycr:
1569         \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`.

```

1570         \dim_gzero_new:N \g_@@_dp_row_zero_dim
1571         \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1572         \dim_gzero_new:N \g_@@_ht_row_zero_dim
1573         \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1574         \dim_gzero_new:N \g_@@_ht_row_one_dim
1575         \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1576         \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1577         \dim_gzero_new:N \g_@@_ht_last_row_dim
1578         \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1579         \dim_gzero_new:N \g_@@_dp_last_row_dim
1580         \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`.

```

1581         \cs_set_eq:NN \ialign \@@_old_ialign:
1582         \halign
1583     }

```

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).

```

1584     \cs_set_eq:NN \@@_old_ldots \ldots
1585     \cs_set_eq:NN \@@_old_cdots \cdots
1586     \cs_set_eq:NN \@@_old_vdots \vdots
1587     \cs_set_eq:NN \@@_old_ddots \ddots

```

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

```

1588 \cs_set_eq:NN \@@_old_iddots \iddots
1589 \bool_if:NTF \l_@@_standard_cline_bool
1590 { \cs_set_eq:NN \cline \@@_standard_cline }
1591 { \cs_set_eq:NN \cline \@@_cline }
1592 \cs_set_eq:NN \Ldots \@@_Ldots
1593 \cs_set_eq:NN \Cdots \@@_Cdots
1594 \cs_set_eq:NN \Vdots \@@_Vdots
1595 \cs_set_eq:NN \Ddots \@@_Ddots
1596 \cs_set_eq:NN \Iddots \@@_Iddots
1597 \cs_set_eq:NN \Hline \@@_Hline:
1598 \cs_set_eq:NN \Hspace \@@_Hspace:
1599 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1600 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1601 \cs_set_eq:NN \Block \@@_Block:
1602 \cs_set_eq:NN \rotate \@@_rotate:
1603 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1604 \cs_set_eq:NN \dotfill \@@_dotfill:
1605 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1606 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1607 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1608 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1609 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1610 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1611 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1612 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1613 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1614 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1615 \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.

```

1616 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1617 \hook_gput_code:nnn { env / tabular / begin } { . }
1618 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1619 \@@_revert_colortbl:

```

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

```

1620 \tl_if_exist:NT \l_@@_note_in_caption_tl
1621 {
1622   \tl_if_empty:NF \l_@@_note_in_caption_tl
1623   {
1624     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1625     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1626   }
1627 }

```

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).

```

1628 \seq_gclear:N \g_@@_multicolumn_cells_seq
1629 \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).

```

1630 \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`).

```
1631 \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.

```
1632 \int_gzero_new:N \g_@@_col_total_int
1633 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1634 \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.

```
1635 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1636 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1637 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1638 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1639 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1640 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1641 \tl_gclear:N \g_nicematrix_code_before_tl
1642 \tl_gclear:N \g_@@_pre_code_before_tl
1643 }
```

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

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

```
1644 \cs_new_protected:Npn \@@_pre_array:
1645 {
1646   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1647   \int_gzero_new:N \c@iRow
1648   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1649   \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).

```
1650 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1651 {
1652   \bool_set_true:N \l_@@_last_row_without_value_bool
1653   \bool_if:NT \g_@@_aux_found_bool
1654   { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1655 }
1656 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1657 {
1658   \bool_if:NT \g_@@_aux_found_bool
1659   { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1660 }
```

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”.

```
1661 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1662 {
1663   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1664   {
1665     \dim_gset:Nn \g_@@_ht_last_row_dim
1666     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1667     \dim_gset:Nn \g_@@_dp_last_row_dim
1668     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1669   }
1670 }
```

```

1671 \seq_gclear:N \g_@@_cols_vlism_seq
1672 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1673 \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.

```

1674 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1675 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1676 \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.

```

1677 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1678 \@@_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.

```

1679 \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]`).

```

1680 \dim_zero_new:N \l_@@_left_delim_dim
1681 \dim_zero_new:N \l_@@_right_delim_dim
1682 \bool_if:NTF \g_@@_delims_bool
1683 {

```

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

```

1684 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1685 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1686 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1687 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1688 }
1689 {
1690 \dim_gset:Nn \l_@@_left_delim_dim
1691 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1692 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1693 }

```

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).

```

1694 \hbox_set:Nw \l_@@_the_array_box
1695 \skip_horizontal:N \l_@@_left_margin_dim
1696 \skip_horizontal:N \l_@@_extra_left_margin_dim
1697 \c_math_toggle_token
1698 \bool_if:NTF \l_@@_light_syntax_bool
1699 { \use:c { @@-light-syntax } }
1700 { \use:c { @@-normal-syntax } }
1701 }

```


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

```

1702 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1703 {
1704   \tl_set:Nn \l_tmpa_tl { #1 }
1705   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1706     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1707   \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1708   \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...

```

1709   \@@_pre_array:
1710 }

```

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).

```

1711 \cs_new_protected:Npn \@@_pre_code_before:
1712 {

```

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).

```

1713   \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1714   \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1715   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1716   \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.

```

1717   \pgfsys@markposition { \@@_env: - position }
1718   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1719   \pgfpicture
1720   \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1721   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1722   {
1723     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1724     \pgfcoordinate { \@@_env: - row - ##1 }
1725     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1726   }

```

Now, the recreation of the `col` nodes.

```

1727   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1728   {
1729     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1730     \pgfcoordinate { \@@_env: - col - ##1 }
1731     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1732   }

```

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

```

1733   \@@_create_diag_nodes:

```

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

```
1734 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1735 \endpgfpicture
```

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

```
1736 \@@_create_blocks_nodes:
1737 \IfPackageLoadedTF { tikz }
1738 {
1739   \tikzset
1740   {
1741     every-picture / .style =
1742     { overlay , name-prefix = \@@_env: - }
1743   }
1744 }
1745 { }
1746 \cs_set_eq:NN \cellcolor \@@_cellcolor
1747 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1748 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1749 \cs_set_eq:NN \rowcolor \@@_rowcolor
1750 \cs_set_eq:NN \rowcolors \@@_rowcolors
1751 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1752 \cs_set_eq:NN \arraycolor \@@_arraycolor
1753 \cs_set_eq:NN \columncolor \@@_columncolor
1754 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1755 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1756 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1757 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1758 }

1759 \cs_new_protected:Npn \@@_exec_code_before:
1760 {
1761   \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`.

```
1762 \@@_add_to_colors_seq:nn { { nocolor } } { }
1763 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1764 \group_begin:
```

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

```
1765 \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`).

```
1766 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1767 { \@@_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`.

```
1768 \exp_last_unbraced:NV \@@_CodeBefore_keys:
1769 \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.

```

1770     \@@_actually_color:
1771     \l_@@_code_before_tl
1772     \q_stop
1773     \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1774     \group_end:
1775     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1776     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1777 }

1778 \keys_define:nn { NiceMatrix / CodeBefore }
1779 {
1780     create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1781     create-cell-nodes .default:n = true ,
1782     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1783     sub-matrix .value_required:n = true ,
1784     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1785     delimiters / color .value_required:n = true ,
1786     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1787 }

1788 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1789 {
1790     \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1791     \@@_CodeBefore:w
1792 }

```

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.

```

1793 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1794 {
1795     \bool_if:NT \g_@@_aux_found_bool
1796     {
1797         \@@_pre_code_before:
1798         #1
1799     }
1800 }

```

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.

```

1801 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1802 {
1803     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1804     {
1805         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1806         \pgfcoordinate { \@@_env: - row - ##1 - base }
1807         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1808         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1809         {
1810             \cs_if_exist:cT
1811             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1812             {
1813                 \pgfsys@getposition
1814                 { \@@_env: - ##1 - #####1 - NW }
1815                 \@@_node_position:
1816                 \pgfsys@getposition
1817                 { \@@_env: - ##1 - #####1 - SE }

```

```

1818         \@@_node_position_i:
1819         \@@_pgf_rect_node:nnn
1820         { \@@_env: - ##1 - #####1 }
1821         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1822         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1823     }
1824 }
1825 }
1826 \int_step_inline:nn \c@iRow
1827 {
1828     \pgfnodealias
1829     { \@@_env: - ##1 - last }
1830     { \@@_env: - ##1 - \int_use:N \c@jCol }
1831 }
1832 \int_step_inline:nn \c@jCol
1833 {
1834     \pgfnodealias
1835     { \@@_env: - last - ##1 }
1836     { \@@_env: - \int_use:N \c@iRow - ##1 }
1837 }
1838 \@@_create_extra_nodes:
1839 }

```

```

1840 \cs_new_protected:Npn \@@_create_blocks_nodes:
1841 {
1842     \pgfpicture
1843     \pgf@relevantforpicturesizefalse
1844     \pgfrememberpicturepositiononpagetrue
1845     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1846     { \@@_create_one_block_node:nnnnn ##1 }
1847     \endpgfpicture
1848 }

```

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

```

1849 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1850 {
1851     \tl_if_empty:nF { #5 }
1852     {
1853         \@@_qpoint:n { col - #2 }
1854         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1855         \@@_qpoint:n { #1 }
1856         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1857         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1858         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1859         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1860         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1861         \@@_pgf_rect_node:nnnnn
1862         { \@@_env: - #5 }
1863         { \dim_use:N \l_tmpa_dim }
1864         { \dim_use:N \l_tmpb_dim }
1865         { \dim_use:N \l_@@_tmpc_dim }
1866         { \dim_use:N \l_@@_tmpd_dim }
1867     }
1868 }

```

```

1869 \cs_new_protected:Npn \@@_patch_for_revtext:
1870 {

```

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).

```

1871 \cs_set_eq:NN \@addamp \@addamp@LaTeX
1872 \cs_set_eq:NN \insert@column \insert@column@array
1873 \cs_set_eq:NN \@classx \@classx@array
1874 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1875 \cs_set_eq:NN \@arraycr \@arraycr@array
1876 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1877 \cs_set_eq:NN \array \array@array
1878 \cs_set_eq:NN \@array \@array@array
1879 \cs_set_eq:NN \@tabular \@tabular@array
1880 \cs_set_eq:NN \@mkpream \@mkpream@array
1881 \cs_set_eq:NN \endarray \endarray@array
1882 \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1883 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1884 }

```

11 The environment `{NiceArrayWithDelims}`

```

1885 \NewDocumentEnvironment { NiceArrayWithDelims }
1886 { m m O { } m ! O { } t \CodeBefore }
1887 {
1888 \bool_if:NT \c_@@_revtex_bool \@_patch_for_revteX:
1889 \@_provide_pgfsyspdfmark:
1890 \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.

```

1891 \bgroup

1892 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1893 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1894 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1895 \int_gzero:N \g_@@_block_box_int
1896 \dim_zero:N \g_@@_width_last_col_dim
1897 \dim_zero:N \g_@@_width_first_col_dim
1898 \bool_gset_false:N \g_@@_row_of_col_done_bool
1899 \str_if_empty:NT \g_@@_name_env_str
1900 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1901 \bool_if:NTF \l_@@_tabular_bool
1902 \mode_leave_vertical:
1903 \@_test_if_math_mode:
1904 \bool_if:NT \l_@@_in_env_bool { \@_fatal:n { Yet~in~env } }
1905 \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.

```

1906 \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.

```

1907 \cs_if_exist:NT \tikz@library@external@loaded
1908 {

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1909      \tikzexternaldisable
1910      \cs_if_exist:NT \ifstandalone
1911      { \tikzset { external / optimize = false } }
1912  }

```

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

```

1913      \int_gincr:N \g_@@_env_int
1914      \bool_if:NF \l_@@_block_auto_columns_width_bool
1915      { \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`).

```

1916      \seq_gclear:N \g_@@_blocks_seq
1917      \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`.

```

1918      \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1919      \seq_gclear:N \g_@@_pos_of_xdots_seq
1920      \tl_gclear_new:N \g_@@_code_before_tl
1921      \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.

```

1922      \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1923      {
1924          \bool_gset_true:N \g_@@_aux_found_bool
1925          \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1926      }
1927      { \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.

```

1928      \tl_build_gbegin:N \g_@@_aux_tl
1929      \tl_if_empty:NF \g_@@_code_before_tl
1930      {
1931          \bool_set_true:N \l_@@_code_before_bool
1932          \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1933      }
1934      \tl_if_empty:NF \g_@@_pre_code_before_tl
1935      { \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`.

```

1936      \bool_if:NTF \g_@@_delims_bool
1937      { \keys_set:nn { NiceMatrix / pNiceArray } }
1938      { \keys_set:nn { NiceMatrix / NiceArray } }
1939      { #3 , #5 }

```

```

1940      \@@_set_CT@arc@:V \l_@@_rules_color_tl

```

The argument `#6` is the last argument of `{NiceArrayWithDelims}`. With that argument of type “`t_ \CodeBefore`”, we test whether there is the keyword `\CodeBefore` at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword `\CodeBefore` and the (other) keyword `\Body`. It’s the job that will do the command `\@@_CodeBefore_Body:w`. After that job, the command `\@@_CodeBefore_Body:w` will go on with `\@@_pre_array:`.

```

1941      \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1942  }

```

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

```

1943  {
1944      \bool_if:NTF \l_@@_light_syntax_bool

```

```

1945     { \use:c { end @@-light-syntax } }
1946     { \use:c { end @@-normal-syntax } }
1947     \c_math_toggle_token
1948     \skip_horizontal:N \l_@@_right_margin_dim
1949     \skip_horizontal:N \l_@@_extra_right_margin_dim
1950     \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.

```

1951     \bool_if:NT \l_@@_width_used_bool
1952     {
1953         \int_if_zero:nT \g_@@_total_X_weight_int
1954         { \@@_error_or_warning:n { width-without-X-columns } }
1955     }

```

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 .

```

1956     \int_compare:nNnT \g_@@_total_X_weight_int > 0
1957     {
1958         \tl_build_gput_right:Nx \g_@@_aux_tl
1959         {
1960             \bool_set_true:N \l_@@_X_columns_aux_bool
1961             \dim_set:Nn \l_@@_X_columns_dim
1962             {
1963                 \dim_compare:nNnTF
1964                 {
1965                     \dim_abs:n
1966                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1967                 }
1968                 <
1969                 { 0.001 pt }
1970                 { \dim_use:N \l_@@_X_columns_dim }
1971                 {
1972                     \dim_eval:n
1973                     {
1974                         ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1975                         / \int_use:N \g_@@_total_X_weight_int
1976                         + \l_@@_X_columns_dim
1977                     }
1978                 }
1979             }
1980         }
1981     }

```

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

```

1982     \int_compare:nNnT \l_@@_last_row_int > { -2 }
1983     {
1984         \bool_if:NF \l_@@_last_row_without_value_bool
1985         {
1986             \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1987             {
1988                 \@@_error:n { Wrong-last-row }
1989                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1990             }
1991         }
1992     }

```

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

```

1993 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1994 \bool_if:nTF \g_@@_last_col_found_bool
1995 { \int_gdecr:N \c@jCol }
1996 {
1997 \int_compare:nNtT \l_@@_last_col_int > { -1 }
1998 { \@@_error:n { last~col~not~used } }
1999 }

```

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

```

2000 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2001 \int_compare:nNtT \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. 88).

```

2002 \int_if_zero:nT \l_@@_first_col_int
2003 {
2004 % \skip_horizontal:N \col@sep % 05-08-23
2005 \skip_horizontal:N \g_@@_width_first_col_dim
2006 }

```

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

```

2007 \bool_if:nTF { ! \g_@@_delims_bool }
2008 {
2009 \str_case:VnF \l_@@_baseline_tl
2010 {
2011 b \@@_use_arraybox_with_notes_b:
2012 c \@@_use_arraybox_with_notes_c:
2013 }
2014 \@@_use_arraybox_with_notes:
2015 }

```

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).

```

2016 {
2017 \int_if_zero:nTF \l_@@_first_row_int
2018 {
2019 \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2020 \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2021 }
2022 { \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

```

2023 \int_compare:nNtTF \l_@@_last_row_int > { -2 }
2024 {
2025 \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2026 \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2027 }
2028 { \dim_zero:N \l_tmpb_dim }
2029 \hbox_set:Nn \l_tmpa_box
2030 {
2031 \c_math_toggle_token
2032 \@@_color:V \l_@@_delimiters_color_tl
2033 \exp_after:wN \left \g_@@_left_delim_tl
2034 \vcenter
2035 {

```

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.

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

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


```

2036         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2037     \hbox
2038     {
2039         \bool_if:NTF \l_@@_tabular_bool
2040         { \skip_horizontal:N -\tabcolsep }
2041         { \skip_horizontal:N -\arraycolsep }
2042         \@@_use_arraybox_with_notes_c:
2043         \bool_if:NTF \l_@@_tabular_bool
2044         { \skip_horizontal:N -\tabcolsep }
2045         { \skip_horizontal:N -\arraycolsep }
2046     }

```

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

```

2047         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
2048     }

```

Curiously, we have to put again the following specification of color. Otherwise, with XeLaTeX (and not with the other engines), the closing delimiter is not colored.

```

2049         \@@_color:V \l_@@_delimiters_color_tl
2050         \exp_after:wN \right \g_@@_right_delim_tl
2051         \c_math_toggle_token
2052     }

```

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.

```

2053         \bool_if:NTF \l_@@_delimiters_max_width_bool
2054         {
2055             \@@_put_box_in_flow_bis:nn
2056             \g_@@_left_delim_tl \g_@@_right_delim_tl
2057         }
2058         \@@_put_box_in_flow:
2059     }

```

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. 89).

```

2060     \bool_if:NT \g_@@_last_col_found_bool
2061     {
2062         \skip_horizontal:N \g_@@_width_last_col_dim
2063         % \skip_horizontal:N \col@sep % 2023-08-05
2064     }
2065     \bool_if:NT \l_@@_preamble_bool
2066     {
2067         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2068         { \@@_warning_gredirect_none:n { columns-not-used } }
2069     }
2070     \@@_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.

```

2071     \egroup

```

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

```

2072     \tl_build_gend:N \g_@@_aux_tl
2073     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2074     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2075     \iow_now:Nx \@mainaux
2076     {
2077         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2078         { \exp_not:V \g_@@_aux_tl }
2079     }
2080     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2081     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2082 }

```

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.

```

2083 \cs_new_protected:Npn \@@_transform_preamble:
2084 {
2085   \@@_transform_preamble_i:
2086   \@@_transform_preamble_ii:
2087 }
2088 \cs_new_protected:Npn \@@_transform_preamble_i:
2089 {
2090   \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 `vlism`).

```

2091   \seq_gclear:N \g_@@_cols_vlism_seq

```

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

```

2092   \bool_gset_false:N \g_tmpb_bool

```

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

```

2093   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2094   \int_zero:N \l_tmpa_int
2095   \tl_gclear:N \g_@@_array_preamble_tl
2096   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2097   {
2098     \tl_gset:Nn \g_@@_array_preamble_tl
2099     { ! { \skip_horizontal:N \arrayrulewidth } }
2100   }
2101   {
2102     \clist_if_in:NnT \l_@@_vlines_clist 1
2103     {
2104       \tl_gset:Nn \g_@@_array_preamble_tl
2105       { ! { \skip_horizontal:N \arrayrulewidth } }
2106     }
2107   }

```

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

```

2108   \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2109   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2110   \@@_replace_columncolor:
2111 }

2112 \hook_gput_code:nnn { begindocument } { . }
2113 {
2114   \IfPackageLoadedTF { colortbl }
2115   {
2116     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2117     \cs_new_protected:Npn \@@_replace_columncolor:

```

```

2118     {
2119         \regex_replace_all:NnN
2120         \c_@@_columncolor_regex
2121         { \c { @@_columncolor_preamble } }
2122         \g_@@_array_preamble_tl
2123     }
2124 }
2125 {
2126     \cs_new_protected:Npn \@@_replace_columncolor:
2127     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2128 }
2129 }

2130 \cs_new_protected:Npn \@@_transform_preamble_ii:
2131 {

```

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

```

2132     \bool_lazy_or:nnT
2133     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
2134     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2135     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2136     \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).

```

2137     \int_if_zero:nTF \l_@@_first_col_int
2138     { \tl_gput_left:NV \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2139     {
2140         \bool_lazy_all:nT
2141         {
2142             { \bool_not_p:n \g_@@_delims_bool }
2143             { \bool_not_p:n \l_@@_tabular_bool }
2144             { \tl_if_empty_p:N \l_@@_vlines_clist }
2145             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2146         }
2147         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2148     }
2149     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2150     { \tl_gput_right:NV \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2151     {
2152         \bool_lazy_all:nT
2153         {
2154             { \bool_not_p:n \g_@@_delims_bool }
2155             { \bool_not_p:n \l_@@_tabular_bool }
2156             { \tl_if_empty_p:N \l_@@_vlines_clist }
2157             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2158         }
2159         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2160     }

```

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`).

```

2161     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2162     {
2163         \tl_gput_right:Nn \g_@@_array_preamble_tl
2164         { > { \@@_error_too_much_cols: } 1 }
2165     }
2166 }

```

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 and

```
2167 \cs_new_protected:Npn \@@_rec_preamble:n #1
2168 {
```

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

```
2169     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2170     { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2171     {
```

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

```
2172     \cs_if_exist:cTF { NC @ find @ #1 }
2173     {
2174         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2175         \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2176     }
2177     {
2178         \tl_if_eq:nnT { #1 } { S }
2179         { \@@_fatal:n { unknown~column~type~S } }
2180         { \@@_fatal:nn { unknown~column~type } { #1 } }
2181     }
2182 }
2183 }
```

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

```
2184 \cs_new:Npn \@@_c #1
2185 {
2186     \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2187     \tl_gclear:N \g_@@_pre_cell_tl
2188     \tl_gput_right:Nn \g_@@_array_preamble_tl
2189     {
2190         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2191         #1
2192         < \@@_cell_end:
2193     }
```

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

```
2194     \int_gincr:N \c@jCol
2195     \@@_rec_preamble_after_col:n
2196 }
2197 \cs_set_eq:NN \@@_l \@@_c
2198 \cs_set_eq:NN \@@_r \@@_c
```

For `!` and `@`

```
2199 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2200 {
2201     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2202     \@@_rec_preamble:n
2203 }
2204 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }
```

For `|`

```
2205 \cs_new:cpn { @@ _ | } #1
2206 {
```

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

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

```

2207 \int_incr:N \l_tmpa_int
2208 \@@_make_preamble_i_i:n
2209 }

2210 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2211 {
2212   \str_if_eq:nnTF { #1 } |
2213   { \use:c { @@ _ | } | }
2214   { \@@_make_preamble_i_ii:nn { } #1 }
2215 }

2216 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2217 {
2218   \str_if_eq:nnTF { #2 } [
2219   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2220   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2221 }

2222 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2223 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2224 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2225 {
2226   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2227   \tl_gput_right:Nx \g_@@_array_preamble_tl
2228   {

```

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

```

2229   \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2230 }
2231 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2232 {
2233   \@@_vline:n
2234   {
2235     position = \int_eval:n { \c@jCol + 1 } ,
2236     multiplicity = \int_use:N \l_tmpa_int ,
2237     total-width = \dim_use:N \l_@@_rule_width_dim ,
2238     #2
2239   }

```

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.

```

2240   }
2241   \int_zero:N \l_tmpa_int
2242   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2243   \@@_rec_preamble:n #1
2244 }

2245 \cs_new:cpn { @@ _ > } #1 #2
2246 {
2247   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2248   \@@_rec_preamble:n
2249 }

2250 \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.

```

2251 \keys_define:nn { WithArrows / p-column }
2252 {
2253   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2254   r .value_forbidden:n = true ,
2255   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2256   c .value_forbidden:n = true ,

```

```

2257 l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2258 l .value_forbidden:n = true ,
2259 R .code:n =
2260   \IfPackageLoadedTF { ragged2e }
2261   { \str_set:Nn \l_@@_hpos_col_str { R } }
2262   {
2263     \@@_error_or_warning:n { ragged2e~not~loaded }
2264     \str_set:Nn \l_@@_hpos_col_str { r }
2265   } ,
2266 R .value_forbidden:n = true ,
2267 L .code:n =
2268   \IfPackageLoadedTF { ragged2e }
2269   { \str_set:Nn \l_@@_hpos_col_str { L } }
2270   {
2271     \@@_error_or_warning:n { ragged2e~not~loaded }
2272     \str_set:Nn \l_@@_hpos_col_str { l }
2273   } ,
2274 L .value_forbidden:n = true ,
2275 C .code:n =
2276   \IfPackageLoadedTF { ragged2e }
2277   { \str_set:Nn \l_@@_hpos_col_str { C } }
2278   {
2279     \@@_error_or_warning:n { ragged2e~not~loaded }
2280     \str_set:Nn \l_@@_hpos_col_str { c }
2281   } ,
2282 C .value_forbidden:n = true ,
2283 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2284 S .value_forbidden:n = true ,
2285 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2286 p .value_forbidden:n = true ,
2287 t .meta:n = p ,
2288 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2289 m .value_forbidden:n = true ,
2290 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2291 b .value_forbidden:n = true ,
2292 }

```

For p, b and m.

```

2293 \cs_new:Npn \@@_p #1
2294 {
2295   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2296   \@@_make_preamble_ii_i:n
2297 }
2298 \cs_set_eq:NN \@@_b \@@_p
2299 \cs_set_eq:NN \@@_m \@@_p

```

```

2300 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2301 {
2302   \str_if_eq:nnTF { #1 } { [ ]
2303     { \@@_make_preamble_ii_ii:w [ ]
2304       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2305     }

```

```

2306 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2307 { \@@_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.

```

2308 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2309 {

```

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).

```

2310 \str_set:Nn \l_@@_hpos_col_str { j }
2311 \tl_set:Nn \l_tmpa_tl { #1 }
2312 \@@_keys_p_column:V \l_tmpa_tl
2313 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2314 }

2315 \cs_new_protected:Npn \@@_keys_p_column:n #1
2316 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2317 \cs_generate_variant:Nn \@@_keys_p_column:n { V }

```

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

```

2318 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2319 {
2320   \use:e
2321   {
2322     \@@_make_preamble_ii_v:nnnnnnnn
2323     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2324     { \dim_eval:n { #1 } }
2325     {

```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_str` which will provide the horizontal alignment of the column to which belongs the cell.

```

2326 \str_if_eq:VnTF \l_@@_hpos_col_str j
2327 { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2328 {
2329   \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2330   { \str_lowercase:V \l_@@_hpos_col_str }
2331 }
2332 \str_case:Vn \l_@@_hpos_col_str
2333 {
2334   c { \exp_not:N \centering }
2335   l { \exp_not:N \raggedright }
2336   r { \exp_not:N \raggedleft }
2337   C { \exp_not:N \Centering }
2338   L { \exp_not:N \RaggedRight }
2339   R { \exp_not:N \RaggedLeft }
2340 }
2341 #3
2342 }
2343 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2344 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2345 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2346 { #2 }
2347 {
2348   \str_case:VnF \l_@@_hpos_col_str
2349   {
2350     { j } { c }
2351     { si } { c }
2352   }

```

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

```

2353 { \str_lowercase:V \l_@@_hpos_col_str }
2354 }
2355 }

```

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

```

2356 \int_gincr:N \c@jCol
2357 \@@_rec_preamble_after_col:n
2358 }

```

#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_str` which will be available in each cell of the column.

#4 is an extra-code which contains `\@@_center_cell_box`: (when the column is a `m` column) or nothing (in the other cases).

#5 is a code put just before the `c` (or `r` or `l`: see #8).

#6 is a code put just after the `c` (or `r` or `l`: see #8).

#7 is the type of environment: `minipage` or `varwidth`.

#8 is the letter `c` or `r` or `l` which is the basic specifier of column which is used *in fine*.

```

2359 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2360 {
2361   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2362     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2363     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2364   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2365   \tl_gclear:N \g_@@_pre_cell_tl
2366   \tl_gput_right:Nn \g_@@_array_preamble_tl
2367     {
2368       > {

```

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.

```

2369       \dim_set:Nn \l_@@_col_width_dim { #2 }
2370       \@@_cell_begin:w

```

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

```

2371       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2372       \everypar
2373       {
2374         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2375         \everypar { }
2376       }

```

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

```

2377       #3

```

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

```

2378       \g_@@_row_style_tl
2379       \arraybackslash
2380       #5
2381     }
2382     #8
2383     < {
2384       #6

```

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

```

2385       \@finalstrut \@arstrutbox
2386       \use:c { end #7 }

```

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

```

2387       #4
2388       \@@_cell_end:
2389     }
2390   }
2391 }

```



```

2392 \str_new:N \c_@@_ignorespaces_str
2393 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2394 \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...

```

2395 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2396 \cs_new_protected:Npn \@@_test_if_empty_i:
2397 {
2398   \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2399   \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2400   { \@@_test_if_empty:w }
2401 }
2402 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2403 {
2404   \peek_meaning:NT \unskip
2405   {
2406     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2407     {
2408       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2409       \skip_horizontal:N \l_@@_col_width_dim
2410     }
2411   }
2412 }
2413 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2414 {
2415   \peek_meaning:NT \__siunitx_table_skip:n
2416   {
2417     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2418     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2419   }
2420 }

```

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.

```

2421 \cs_new_protected:Npn \@@_center_cell_box:
2422 {

```

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

```

2423   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2424   {
2425     \int_compare:nNnT
2426     { \box_ht:N \l_@@_cell_box }
2427     >

```

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).

```

2428     { \box_ht:N \strutbox }
2429     {
2430       \hbox_set:Nn \l_@@_cell_box
2431       {
2432         \box_move_down:nn
2433         {
2434           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2435             + \baselineskip ) / 2
2436         }
2437         { \box_use:N \l_@@_cell_box }
2438     }
2439 }
2440 }
2441 }

```

For V (similar to the V of varwidth).

```

2442 \cs_new:Npn \@@_V #1 #2
2443 {
2444     \str_if_eq:nnTF { #2 } { [ ] }
2445     { \@@_make_preamble_V_i:w [ ] }
2446     { \@@_make_preamble_V_i:w [ ] { #2 } }
2447 }
2448 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2449 { \@@_make_preamble_V_ii:nn { #1 } }
2450 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2451 {
2452     \str_set:Nn \l_@@_vpos_col_str { p }
2453     \str_set:Nn \l_@@_hpos_col_str { j }
2454     \tl_set:Nn \l_tmpa_tl { #1 }
2455     \@@_keys_p_column:V \l_tmpa_tl
2456     \IfPackageLoadedTF { varwidth }
2457     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2458     {
2459         \@@_error_or_warning:n { varwidth-not-loaded }
2460         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2461     }
2462 }

```

For w and W

```

2463 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2464 \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.

```

2465 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2466 {
2467     \str_if_eq:nnTF { #3 } { s }
2468     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2469     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2470 }

```

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.

```

2471 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2472 {
2473     \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2474     \tl_gclear:N \g_@@_pre_cell_tl
2475     \tl_gput_right:Nn \g_@@_array_preamble_tl
2476     {
2477         > {
2478             \dim_set:Nn \l_@@_col_width_dim { #2 }
2479             \@@_cell_begin:w
2480             \str_set:Nn \l_@@_hpos_cell_str { c }
2481         }
2482         c
2483         < {

```

```

2484         \@@_cell_end_for_w_s:
2485         #1
2486         \@@_adjust_size_box:
2487         \box_use_drop:N \l_@@_cell_box
2488     }
2489 }
2490 \int_gincr:N \c@jCol
2491 \@@_rec_preamble_after_col:n
2492 }

```

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

```

2493 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2494 {
2495     \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2496     \tl_gclear:N \g_@@_pre_cell_tl
2497     \tl_gput_right:Nn \g_@@_array_preamble_tl
2498     {
2499         > {

```

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.

```

2500         \dim_set:Nn \l_@@_col_width_dim { #4 }
2501         \hbox_set:Nw \l_@@_cell_box
2502         \@@_cell_begin:w
2503         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2504     }
2505     c
2506     < {
2507         \@@_cell_end:
2508         \hbox_set_end:
2509         #1
2510         \@@_adjust_size_box:
2511         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2512     }
2513 }

```

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

```

2514     \int_gincr:N \c@jCol
2515     \@@_rec_preamble_after_col:n
2516 }

```

```

2517 \cs_new_protected:Npn \@@_special_W:
2518 {
2519     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2520     { \@@_warning:n { W~warning } }
2521 }

```

For S (of siunitx).

```

2522 \cs_new:Npn \@@_S #1 #2
2523 {
2524     \str_if_eq:nnTF { #2 } { [ ] }
2525     { \@@_make_preamble_S:w [ ] }
2526     { \@@_make_preamble_S:w [ ] { #2 } }
2527 }
2528 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2529 { \@@_make_preamble_S_i:n { #1 } }
2530 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2531 {
2532     \IfPackageLoadedTF { siunitx }
2533     {
2534         \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2535         \tl_gclear:N \g_@@_pre_cell_tl

```

```

2536 \tl_gput_right:Nn \g_@@_array_preamble_tl
2537 {
2538   > {
2539     \@@_cell_begin:w
2540     \keys_set:nn { siunitx } { #1 }
2541     \siunitx_cell_begin:w
2542   }
2543   c
2544   < { \siunitx_cell_end: \@@_cell_end: }
2545 }

```

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

```

2546 \int_gincr:N \c@jCol
2547 \@@_rec_preamble_after_col:n
2548 }
2549 { \@@_fatal:n { siunitx-not~loaded } }
2550 }

```

For (, [and \{.

```

2551 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2552 {
2553   \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.

```

2554 \int_if_zero:nTF \c@jCol
2555 {
2556   \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2557   {

```

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

```

2558 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2559 \tl_gset:Nn \g_@@_right_delim_tl { . }
2560 \@@_rec_preamble:n #2
2561 }
2562 {
2563   \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2564   \@@_make_preamble_iv:nn { #1 } { #2 }
2565 }
2566 }
2567 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2568 }
2569 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2570 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2571 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2572 {
2573   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2574   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2575   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2576   {
2577     \@@_error:nn { delimiter~after~opening } { #2 }
2578     \@@_rec_preamble:n
2579   }
2580   { \@@_rec_preamble:n #2 }
2581 }

```

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

```

2582 \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}).

```

2583 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2584 {
2585   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2586   \tl_if_in:nnTF { ) ] \} } { #2 }
2587   { \@@_make_preamble_v:nnn #1 #2 }
2588   {
2589     \tl_if_eq:nnTF { \stop } { #2 }
2590     {
2591       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2592       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2593       {
2594         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2595         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2596           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2597         \@@_rec_preamble:n #2
2598       }
2599     }
2600     {
2601       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2602       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2603       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2604         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2605       \@@_rec_preamble:n #2
2606     }
2607   }
2608 }
2609 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N } }
2610 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2611 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2612 {
2613   \tl_if_eq:nnTF { \stop } { #3 }
2614   {
2615     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2616     {
2617       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2618       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2619         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2620       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2621     }
2622     {
2623       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2624       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2625         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2626       \@@_error:nn { double~closing~delimiter } { #2 }
2627     }
2628   }
2629   {
2630     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2631       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2632     \@@_error:nn { double~closing~delimiter } { #2 }
2633     \@@_rec_preamble:n #3
2634   }
2635 }
2636 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2637 { \use:c { @@ _ \token_to_str:N ) } }

```

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

```

2638 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2639 {

```

```

2640 \str_if_eq:nnTF { #1 } { < }
2641 \@@_rec_preamble_after_col_i:n
2642 {
2643   \str_if_eq:nnTF { #1 } { @ }
2644   \@@_rec_preamble_after_col_ii:n
2645   {
2646     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2647     {
2648       \tl_gput_right:Nn \g_@@_array_preamble_tl
2649       { ! { \skip_horizontal:N \arrayrulewidth } }
2650     }
2651     {
2652       \exp_args:NNe
2653       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2654       {
2655         \tl_gput_right:Nn \g_@@_array_preamble_tl
2656         { ! { \skip_horizontal:N \arrayrulewidth } }
2657       }
2658     }
2659     \@@_rec_preamble:n { #1 }
2660   }
2661 }
2662 }
2663 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2664 {
2665   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2666   \@@_rec_preamble_after_col:n
2667 }

```

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.

```

2668 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2669 {
2670   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2671   {
2672     \tl_gput_right:Nn \g_@@_array_preamble_tl
2673     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2674   }
2675   {
2676     \exp_args:NNe
2677     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2678     {
2679       \tl_gput_right:Nn \g_@@_array_preamble_tl
2680       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2681     }
2682     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2683   }
2684   \@@_rec_preamble:n
2685 }

```

```

2686 \cs_new:cpn { @@ _ * } #1 #2 #3
2687 {
2688   \tl_clear:N \l_tmpa_tl
2689   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2690   \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpa_tl
2691 }

```

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.

```

2692 \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.

```

2693 \cs_new:Npn \@@_X #1 #2
2694 {
2695   \str_if_eq:nnTF { #2 } { [ ]
2696     { \@@_make_preamble_X:w [ ]
2697       { \@@_make_preamble_X:w [ ] #2 }
2698     }
2699 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2700 { \@@_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`).

```

2701 \keys_define:nn { WithArrows / X-column }
2702 { 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.

```

2703 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2704 {

```

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).

```

2705   \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).

```

2706   \tl_set:Nn \l_@@_vpos_col_str { p }

```

The integer `\l_@@_weight_int` will be the weight of the X column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the X columns are used in the computation of the actual width of those columns as in `tabu` (now obsolete) or `tabularray`.

```

2707   \int_zero_new:N \l_@@_weight_int
2708   \int_set:Nn \l_@@_weight_int { 1 }
2709   \tl_set:Nn \l_tmpa_tl { #1 }
2710   \@@_keys_p_column:V \l_tmpa_tl
2711   \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2712   \int_compare:nNnT \l_@@_weight_int < 0
2713   {
2714     \@@_error_or_warning:n { negative-weight }
2715     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2716   }
2717   \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).

```

2718   \bool_if:NTF \l_@@_X_columns_aux_bool
2719   {
2720     \exp_args:Nne
2721     \@@_make_preamble_ii_iv:nnn
2722     { \l_@@_weight_int \l_@@_X_columns_dim }
2723     { minipage }
2724     { \@@_no_update_width: }
2725   }
2726   {
2727     \tl_gput_right:Nn \g_@@_array_preamble_tl
2728     {
2729       > {
2730         \@@_cell_begin:w
2731         \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`.

```
2732 \NotEmpty
```

The following code will nullify the box of the cell.

```
2733 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2734 { \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`.

```
2735 \begin { minipage } { 5 cm } \arraybackslash
2736 }
2737 c
2738 < {
2739 \end { minipage }
2740 \@@_cell_end:
2741 }
2742 }
2743 \int_gincr:N \c@jCol
2744 \@@_rec_preamble_after_col:n
2745 }
2746 }

2747 \cs_new_protected:Npn \@@_no_update_width:
2748 {
2749 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2750 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2751 }
```

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

```
2752 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2753 {
2754 \seq_gput_right:Nx \g_@@_cols_vlism_seq
2755 { \int_eval:n { \c@jCol + 1 } }
2756 \tl_gput_right:Nx \g_@@_array_preamble_tl
2757 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2758 \@@_rec_preamble:n
2759 }
```

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.

```
2760 \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).

```
2761 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2762 { \@@_fatal:n { Preamble~forgotten } }
2763 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2764 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2765 \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).

```
2766 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2767 {
```


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 that one column specifier in the preamble of `\multicolumn`.

```

2768 \multispan { #1 }
2769 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2770 \begingroup
2771 \cs_set:Npn \@addamp
2772 { \legacy_if:nTF { \@firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2773 \tl_gclear:N \g_@@_preamble_tl
2774 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2775 \exp_args:NV \@mkpream \g_@@_preamble_tl
2776 \@addtopreamble \@empty
2777 \endgroup

```

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

```

2778 \int_compare:nNnT { #1 } > 1
2779 {
2780   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2781   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2782   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2783   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2784   {
2785     {
2786       \int_if_zero:nTF \c@jCol
2787       { \int_eval:n { \c@iRow + 1 } }
2788       { \int_use:N \c@iRow }
2789     }
2790     { \int_eval:n { \c@jCol + 1 } }
2791     {
2792       \int_if_zero:nTF \c@jCol
2793       { \int_eval:n { \c@iRow + 1 } }
2794       { \int_use:N \c@iRow }
2795     }
2796     { \int_eval:n { \c@jCol + #1 } }
2797     { } % for the name of the block
2798   }
2799 }

```

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

```

2800 \cs_set:Npn \@sharp { #3 }
2801 \@arstrut
2802 \@preamble
2803 \null

```

We add some lines.

```

2804 \int_gadd:Nn \c@jCol { #1 - 1 }
2805 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2806 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2807 \ignorespaces
2808 }

```

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`.

```

2809 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2810 {
2811   \str_case:nnF { #1 }
2812   {

```

```

2813 c { \@@_make_m_preamble_i:n #1 }
2814 l { \@@_make_m_preamble_i:n #1 }
2815 r { \@@_make_m_preamble_i:n #1 }
2816 > { \@@_make_m_preamble_ii:nn #1 }
2817 ! { \@@_make_m_preamble_ii:nn #1 }
2818 @ { \@@_make_m_preamble_ii:nn #1 }
2819 | { \@@_make_m_preamble_iii:n #1 }
2820 p { \@@_make_m_preamble_iv:nnn t #1 }
2821 m { \@@_make_m_preamble_iv:nnn c #1 }
2822 b { \@@_make_m_preamble_iv:nnn b #1 }
2823 w { \@@_make_m_preamble_v:nnnn { } #1 }
2824 W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2825 \q_stop { }
2826 }
2827 {
2828 \cs_if_exist:cTF { NC @ find @ #1 }
2829 {
2830 \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2831 \exp_last_unbraced:NV \@@_make_m_preamble:n \l_tmpa_tl
2832 }
2833 {
2834 \tl_if_eq:nnT { #1 } { S }
2835 { \@@_fatal:n { unknown~column~type~S } }
2836 { \@@_fatal:nn { unknown~column~type } { #1 } }
2837 }
2838 }
2839 }

```

For c, l and r

```

2840 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2841 {
2842 \tl_gput_right:Nn \g_@@_preamble_tl
2843 {
2844 > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2845 #1
2846 < \@@_cell_end:
2847 }

```

We test for the presence of a <.

```

2848 \@@_make_m_preamble_x:n
2849 }

```

For >, ! and @

```

2850 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2851 {
2852 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2853 \@@_make_m_preamble:n
2854 }

```

For |

```

2855 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2856 {
2857 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2858 \@@_make_m_preamble:n
2859 }

```

For p, m and b

```

2860 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2861 {
2862 \tl_gput_right:Nn \g_@@_preamble_tl
2863 {
2864 > {
2865 \@@_cell_begin:w

```

```

2866         \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2867         \mode_leave_vertical:
2868         \arraybackslash
2869         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2870     }
2871     c
2872     < {
2873         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2874         \end { minipage }
2875         \@@_cell_end:
2876     }
2877 }

```

We test for the presence of a <.

```

2878     \@@_make_m_preamble_x:n
2879 }

```

For w and W

```

2880 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2881 {
2882     \tl_gput_right:Nn \g_@@_preamble_tl
2883     {
2884         > {
2885             \dim_set:Nn \l_@@_col_width_dim { #4 }
2886             \hbox_set:Nw \l_@@_cell_box
2887             \@@_cell_begin:w
2888             \str_set:Nn \l_@@_hpos_cell_str { #3 }
2889         }
2890         c
2891         < {
2892             \@@_cell_end:
2893             \hbox_set_end:
2894             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2895             #1
2896             \@@_adjust_size_box:
2897             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2898         }
2899     }

```

We test for the presence of a <.

```

2900     \@@_make_m_preamble_x:n
2901 }

```

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

```

2902 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2903 {
2904     \str_if_eq:nnTF { #1 } { < }
2905     \@@_make_m_preamble_ix:n
2906     { \@@_make_m_preamble:n { #1 } }
2907 }
2908 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2909 {
2910     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2911     \@@_make_m_preamble_x:n
2912 }

```

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).

```

2913 \cs_new_protected:Npn \@@_put_box_in_flow:
2914 {
2915     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }

```

```

2916 \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2917 \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2918 { \box_use_drop:N \l_tmpa_box }
2919 \@@_put_box_in_flow_i:
2920 }

```

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).

```

2921 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2922 {
2923   \pgfpicture
2924   \@@_qpoint:n { row - 1 }
2925   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2926   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2927   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2928   \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).

```

2929 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2930 {
2931   \int_set:Nn \l_tmpa_int
2932   {
2933     \str_range:Nnn
2934     \l_@@_baseline_tl
2935     6
2936     { \tl_count:V \l_@@_baseline_tl }
2937   }
2938   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2939 }
2940 {
2941   \str_case:VnF \l_@@_baseline_tl
2942   {
2943     { t } { \int_set:Nn \l_tmpa_int 1 }
2944     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2945   }
2946   { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2947   \bool_lazy_or:nnT
2948   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2949   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2950   {
2951     \@@_error:n { bad-value-for-baseline }
2952     \int_set:Nn \l_tmpa_int 1
2953   }
2954   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2955 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2956 }
2957 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2958 \endpgfpicture
2959 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2960 \box_use_drop:N \l_tmpa_box
2961 }

```

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).

```

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

```

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.

```

2964 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2965 {
2966   \int_compare:nNnT \c@jCol > 1 % added 2023-08-13
2967   {
2968     \box_set_wd:Nn \l_@@_the_array_box
2969     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2970   }
2971 }

```

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

```

2972 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2973 \bool_if:NT \l_@@_caption_above_bool
2974 {
2975   \tl_if_empty:NF \l_@@_caption_tl
2976   {
2977     \bool_set_false:N \g_@@_caption_finished_bool
2978     \int_gzero:N \c@tabularnote
2979     \@@_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).

```

2980 \int_compare:nNnT \g_@@_notes_caption_int > 0
2981 {
2982   \tl_build_gput_right:Nx \g_@@_aux_tl
2983   {
2984     \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2985     { \int_use:N \g_@@_notes_caption_int }
2986   }
2987   \int_gzero:N \g_@@_notes_caption_int
2988 }
2989 }
2990 }

```

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

```

2991 \hbox
2992 {
2993   \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.

```

2994 \@@_create_extra_nodes:
2995 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2996 }

```

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

```

2997 \bool_lazy_any:nT
2998 {
2999   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3000   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3001   { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
3002 }
3003 \@@_insert_tabularnotes:
3004 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3005 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:

```

```

3006   \end { minipage }
3007 }

3008 \cs_new_protected:Npn \@@_insert_caption:
3009 {
3010   \tl_if_empty:NF \l_@@_caption_tl
3011   {
3012     \cs_if_exist:NTF \@capttype
3013     { \@@_insert_caption_i: }
3014     { \@@_error:n { caption~outside~float } }
3015   }
3016 }

```

```

3017 \cs_new_protected:Npn \@@_insert_caption_i:
3018 {
3019   \group_begin:

```

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

```

3020   \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.

```

3021   \IfPackageLoadedTF { floatrow }
3022   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3023   { }
3024   \tl_if_empty:NTF \l_@@_short_caption_tl
3025   { \caption }
3026   { \caption [ \l_@@_short_caption_tl ] }
3027   { \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.

```

3028   \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
3029   {
3030     \bool_gset_true:N \g_@@_caption_finished_bool
3031     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3032     \int_gzero:N \c@tabularnote
3033   }
3034   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3035   \group_end:
3036 }

```

```

3037 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3038 {
3039   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3040   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3041 }

3042 \cs_new_protected:Npn \@@_insert_tabularnotes:
3043 {
3044   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3045   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3046   \skip_vertical:N 0.65ex

```

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

```

3047   \group_begin:
3048   \l_@@_notes_code_before_tl
3049   \tl_if_empty:NF \g_@@_tabularnote_tl
3050   {

```

```

3051      \g_@@_tabularnote_tl \par
3052      \tl_gclear:N \g_@@_tabularnote_tl
3053    }

```

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.

```

3054      \int_compare:nNnT \c@tabularnote > 0
3055      {
3056        \bool_if:NTF \l_@@_notes_para_bool
3057        {
3058          \begin { tabularnotes* }
3059            \seq_map_inline:Nn \g_@@_notes_seq
3060              { \@@_one_tabularnote:nn ##1 }
3061            \strut
3062          \end { tabularnotes* }

```

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

```

3063      \par
3064    }
3065    {
3066      \tabularnotes
3067        \seq_map_inline:Nn \g_@@_notes_seq
3068          { \@@_one_tabularnote:nn ##1 }
3069        \strut
3070      \endtabularnotes
3071    }
3072  }
3073  \unskip
3074  \group_end:
3075  \bool_if:NT \l_@@_notes_bottomrule_bool
3076  {
3077    \IfPackageLoadedTF { booktabs }
3078    {

```

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

```

3079      \skip_vertical:N \aboverulesep

```

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

```

3080      { \CT@arc@ \hrule height \heavyrulewidth }
3081    }
3082    { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3083  }
3084  \l_@@_notes_code_after_tl
3085  \seq_gclear:N \g_@@_notes_seq
3086  \seq_gclear:N \g_@@_notes_in_caption_seq
3087  \int_gzero:N \c@tabularnote
3088 }

```

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

```

3089 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3090 {
3091   \tl_if_novalue:nTF { #1 }
3092   { \item }
3093   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3094 }

```

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`.

```

3095 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:

```

```

3096 {
3097   \pgfpicture
3098     \@@_qpoint:n { row - 1 }
3099     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3100     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3101     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3102   \endpgfpicture
3103   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3104   \int_if_zero:nT \l_@@_first_row_int
3105   {
3106     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3107     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3108   }
3109   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3110 }

```

Now, the general case.

```

3111 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3112 {

```

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

```

3113   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3114   { \tl_set:Nn \l_@@_baseline_tl { 1 } }

```

Now, we convert the value of `\l_@@_baseline_tl` (which should represent an integer) to an integer stored in `\l_tmpa_int`.

```

3115   \pgfpicture
3116   \@@_qpoint:n { row - 1 }
3117   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3118   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3119   {
3120     \int_set:Nn \l_tmpa_int
3121     {
3122       \str_range:Nnn
3123         \l_@@_baseline_tl
3124         6
3125       { \tl_count:V \l_@@_baseline_tl }
3126     }
3127     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3128   }
3129   {
3130     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3131     \bool_lazy_or:nnT
3132     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3133     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3134     {
3135       \@@_error:n { bad~value~for~baseline }
3136       \int_set:Nn \l_tmpa_int 1
3137     }
3138     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3139   }
3140   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3141   \endpgfpicture
3142   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3143   \int_if_zero:nT \l_@@_first_row_int
3144   {
3145     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3146     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3147   }
3148   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3149 }

```

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.

```
3150 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3151 {
```

We will compute the real width of both delimiters used.

```
3152 \dim_zero_new:N \l_@@_real_left_delim_dim
3153 \dim_zero_new:N \l_@@_real_right_delim_dim
3154 \hbox_set:Nn \l_tmpb_box
3155 {
3156   \c_math_toggle_token
3157   \left #1
3158   \vcenter
3159   {
3160     \vbox_to_ht:nn
3161     { \box_ht_plus_dp:N \l_tmpa_box }
3162     { }
3163   }
3164   \right .
3165   \c_math_toggle_token
3166 }
3167 \dim_set:Nn \l_@@_real_left_delim_dim
3168 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3169 \hbox_set:Nn \l_tmpb_box
3170 {
3171   \c_math_toggle_token
3172   \left .
3173   \vbox_to_ht:nn
3174   { \box_ht_plus_dp:N \l_tmpa_box }
3175   { }
3176   \right #2
3177   \c_math_toggle_token
3178 }
3179 \dim_set:Nn \l_@@_real_right_delim_dim
3180 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3181 \skip_horizontal:N \l_@@_left_delim_dim
3182 \skip_horizontal:N -\l_@@_real_left_delim_dim
3183 \@@_put_box_in_flow:
3184 \skip_horizontal:N \l_@@_right_delim_dim
3185 \skip_horizontal:N -\l_@@_real_right_delim_dim
3186 }
```

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).

```
3187 \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).

```
3188 {
3189   \peek_remove_spaces:n
3190   {
3191     \peek_meaning:NTF \end
3192     \@@_analyze_end:Nn
3193     {
3194       \@@_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`).

```

3195         \exp_args:NV \@@_array: \g_@@_array_preamble_tl
3196     }
3197 }
3198 }
3199 {
3200     \@@_create_col_nodes:
3201     \endarray
3202 }
```

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

```

3203 \NewDocumentEnvironment { @@-light-syntax } { b }
3204 {
```

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`.

```

3205     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3206     \tl_map_inline:nn { #1 }
3207     {
3208         \str_if_eq:nnT { ##1 } { & }
3209         { \@@_fatal:n { ampersand-in-light-syntax } }
3210         \str_if_eq:nnT { ##1 } { \ }
3211         { \@@_fatal:n { double-backslash-in-light-syntax } }
3212     }
```

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`.

```

3213     \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```

3214 }
```

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.

```

3215 {
3216     \@@_create_col_nodes:
3217     \endarray
3218 }
3219 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3220 {
3221     \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).

```

3222     \seq_clear_new:N \l_@@_rows_seq
```

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

```

3223     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3224     \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```

3225     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3226     \tl_if_empty:NF \l_tmpa_tl
3227     { \seq_put_right:NV \l_@@_rows_seq \l_tmpa_tl }
```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to compute that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```
3228 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3229 { \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`).

```
3230 \tl_build_begin:N \l_@@_new_body_tl
3231 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3232 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3233 \@@_line_with_light_syntax:V \l_tmpa_tl
```

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

```
3234 \seq_map_inline:Nn \l_@@_rows_seq
3235 {
3236   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3237   \@@_line_with_light_syntax:n { #1 }
3238 }
3239 \tl_build_end:N \l_@@_new_body_tl
3240 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3241 {
3242   \int_set:Nn \l_@@_last_col_int
3243   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3244 }
```

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.

```
3245 \@@_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`).

```
3246 \exp_args:NV \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3247 }
3248 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3249 {
3250   \seq_clear_new:N \l_@@_cells_seq
3251   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3252   \int_set:Nn \l_@@_nb_cols_int
3253   {
3254     \int_max:nn
3255     \l_@@_nb_cols_int
3256     { \seq_count:N \l_@@_cells_seq }
3257   }
3258   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3259   \exp_args:NNV \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3260   \seq_map_inline:Nn \l_@@_cells_seq
3261   { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3262 }
3263 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { V }
```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security). When this command is used, `#1` is, in fact, always `\end`.

```
3264 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3265 {
3266   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3267   { \@@_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.

```
3268   \end { #2 }
3269 }
```

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).

```
3270 \cs_new:Npn \@@_create_col_nodes:
3271 {
3272   \crr
3273   \int_if_zero:nT \l_@@_first_col_int
3274   {
3275     \omit
3276     \hbox_overlap_left:n
3277     {
3278       \bool_if:NT \l_@@_code_before_bool
3279       { \pgfsys@markposition { \@@_env: - col - 0 } }
3280       \pgfpicture
3281       \pgfrememberpicturepositiononpagetrue
3282       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3283       \str_if_empty:NF \l_@@_name_str
3284       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3285       \endpgfpicture
3286       \skip_horizontal:N 2\col@sep
3287       \skip_horizontal:N \g_@@_width_first_col_dim
3288     }
3289     &
3290   }
3291   \omit
```

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

```
3292   \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`).

```
3293   \int_if_zero:nTF \l_@@_first_col_int
3294   {
3295     \bool_if:NT \l_@@_code_before_bool
3296     {
3297       \hbox
3298       {
3299         \skip_horizontal:N -0.5\arrayrulewidth
3300         \pgfsys@markposition { \@@_env: - col - 1 }
3301         \skip_horizontal:N 0.5\arrayrulewidth
3302       }
3303     }
3304     \pgfpicture
3305     \pgfrememberpicturepositiononpagetrue
3306     \pgfcoordinate { \@@_env: - col - 1 }
3307     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3308     \str_if_empty:NF \l_@@_name_str
3309     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3310     \endpgfpicture
3311   }
3312   {
3313     \bool_if:NT \l_@@_code_before_bool
3314     {
3315       \hbox
3316       {
3317         \skip_horizontal:N 0.5\arrayrulewidth
3318         \pgfsys@markposition { \@@_env: - col - 1 }
3319         \skip_horizontal:N -0.5\arrayrulewidth
```

```

3320     }
3321   }
3322   \pgfpicture
3323   \pgfrememberpicturepositiononpagetrue
3324   \pgfcoordinate { \@@_env: - col - 1 }
3325   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3326   \str_if_empty:NF \l_@@_name_str
3327   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3328   \endpgfpicture
3329 }

```

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\lfill`) but we will add some dimensions to it.

```

3330   \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3331   \bool_if:NF \l_@@_auto_columns_width_bool
3332   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3333   {
3334     \bool_lazy_and:nnTF
3335       \l_@@_auto_columns_width_bool
3336       { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3337       { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3338       { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3339     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3340   }
3341   \skip_horizontal:N \g_tmpa_skip
3342   \hbox
3343   {
3344     \bool_if:NT \l_@@_code_before_bool
3345     {
3346       \hbox
3347       {
3348         \skip_horizontal:N -0.5\arrayrulewidth
3349         \pgfsys@markposition { \@@_env: - col - 2 }
3350         \skip_horizontal:N 0.5\arrayrulewidth
3351       }
3352     }
3353     \pgfpicture
3354     \pgfrememberpicturepositiononpagetrue
3355     \pgfcoordinate { \@@_env: - col - 2 }
3356     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3357     \str_if_empty:NF \l_@@_name_str
3358     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3359     \endpgfpicture
3360   }

```

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.

```

3361   \int_gset:Nn \g_tmpa_int 1
3362   \bool_if:NTF \g_@@_last_col_found_bool
3363   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3364   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3365   {
3366     &
3367     \omit
3368     \int_gincr:N \g_tmpa_int

```

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

```

3369     \skip_horizontal:N \g_tmpa_skip
3370     \bool_if:NT \l_@@_code_before_bool
3371     {

```

```

3372         \hbox
3373         {
3374             \skip_horizontal:N -0.5\arrayrulewidth
3375             \pgfsys@markposition
3376             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3377             \skip_horizontal:N 0.5\arrayrulewidth
3378         }
3379     }

```

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

```

3380     \pgfpicture
3381     \pgfrememberpicturepositiononpagetrue
3382     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3383     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3384     \str_if_empty:NF \l_@@_name_str
3385     {
3386         \pgfnodealias
3387         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3388         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3389     }
3390     \endpgfpicture
3391 }

```

```

3392 &
3393 \omit

```

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

```

3394 \int_compare:nNnT \g_@@_col_total_int = 1
3395 { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3396 \skip_horizontal:N \g_tmpa_skip
3397 \int_gincr:N \g_tmpa_int
3398 \bool_lazy_all:nT
3399 {
3400     { \bool_not_p:n \g_@@_delims_bool }
3401     { \bool_not_p:n \l_@@_tabular_bool }
3402     { \clist_if_empty_p:N \l_@@_vlines_clist }
3403     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3404     { ! \l_@@_bar_at_end_of_pream_bool }
3405 }
3406 { \skip_horizontal:N -\col@sep }
3407 \bool_if:NT \l_@@_code_before_bool
3408 {
3409     \hbox
3410     {
3411         \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.

```

3412         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3413         { \skip_horizontal:N -\arraycolsep }
3414         \pgfsys@markposition
3415         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3416         \skip_horizontal:N 0.5\arrayrulewidth
3417         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3418         { \skip_horizontal:N \arraycolsep }
3419     }
3420 }
3421 \pgfpicture
3422 \pgfrememberpicturepositiononpagetrue
3423 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3424 {
3425     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3426         {
3427             \pgfpoint
3428             { - 0.5 \arrayrulewidth - \arraycolsep }
3429             \c_zero_dim
3430         }
3431         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3432     }
3433     \str_if_empty:NF \l_@@_name_str
3434     {
3435         \pgfnodealias
3436         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3437         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3438     }
3439     \endpgfpicture

3440     \bool_if:NT \g_@@_last_col_found_bool
3441     {
3442         \hbox_overlap_right:n
3443         {
3444             \skip_horizontal:N \g_@@_width_last_col_dim
3445             \skip_horizontal:N \col@sep % added 2023-11-05
3446             \bool_if:NT \l_@@_code_before_bool
3447             {
3448                 \pgfsys@markposition
3449                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3450             }
3451             \pgfpicture
3452             \pgfrememberpicturepositiononpagetrue
3453             \pgfcoordinate
3454             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3455             \pgfpointorigin
3456             \str_if_empty:NF \l_@@_name_str
3457             {
3458                 \pgfnodealias
3459                 {
3460                     \l_@@_name_str - col
3461                     - \int_eval:n { \g_@@_col_total_int + 1 }
3462                 }
3463                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3464             }
3465             \endpgfpicture
3466         }
3467     }
3468     \cr
3469 }

```

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

```

3470 \tl_const:Nn \c_@@_preamble_first_col_tl
3471 {
3472     >
3473     {

```

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).

```

3474     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3475     \bool_gset_true:N \g_@@_after_col_zero_bool
3476     \@@_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.

```

3477     \hbox_set:Nw \l_@@_cell_box
3478     \@@_math_toggle_token:
3479     \bool_if:NT \l_@@_small_bool \scriptstyle

```

We insert `\l_@@_code_for_first_col_tl...` but we don't insert it in the potential “first row” and in the potential “last row”.

```

3480     \bool_lazy_and:nnT
3481     { \int_compare_p:nNn \c@iRow > 0 }
3482     {
3483       \bool_lazy_or_p:nn
3484       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3485       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3486     }
3487     {
3488       \l_@@_code_for_first_col_tl
3489       \xglobal \colorlet { nicematrix-first-col } { . }
3490     }
3491   }

```

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`.

```

3492     l
3493     <
3494     {
3495       \@@_math_toggle_token:
3496       \hbox_set_end:
3497       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3498       \@@_adjust_size_box:
3499       \@@_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.

```

3500     \dim_gset:Nn \g_@@_width_first_col_dim
3501     { \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.

```

3502     \hbox_overlap_left:n
3503     {
3504       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3505       \@@_node_for_cell:
3506       { \box_use_drop:N \l_@@_cell_box }
3507       \skip_horizontal:N \l_@@_left_delim_dim
3508       \skip_horizontal:N \l_@@_left_margin_dim
3509       \skip_horizontal:N \l_@@_extra_left_margin_dim
3510     }
3511     \bool_gset_false:N \g_@@_empty_cell_bool
3512     \skip_horizontal:N -2\col@sep
3513   }
3514 }

```

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

```

3515 \tl_const:Nn \c_@@_preamble_last_col_tl
3516 {
3517   >
3518   {
3519     \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).

```

3520     \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.

```

3521     \bool_gset_true:N \g_@@_last_col_found_bool
3522     \int_gincr:N \c@jCol
3523     \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.

```

3524     \hbox_set:Nw \l_@@_cell_box
3525     \@@_math_toggle_token:

```



```
3526 \bool_if:NT \l_@@_small_bool \scriptstyle
```

We insert `\l_@@_code_for_last_col_tl...` but we don't insert it in the potential “first row” and in the potential “last row”.

```
3527 \int_compare:nNnT \c@iRow > 0
3528 {
3529   \bool_lazy_or:nnT
3530   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3531   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3532   {
3533     \l_@@_code_for_last_col_tl
3534     \xglobal \colorlet { nicematrix-last-col } { . }
3535   }
3536 }
3537 }
3538 l
3539 <
3540 {
3541   \@@_math_toggle_token:
3542   \hbox_set_end:
3543   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3544   \@@_adjust_size_box:
3545   \@@_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.

```
3546 \dim_gset:Nn \g_@@_width_last_col_dim
3547 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3548 \skip_horizontal:N -2\col@sep
```

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

```
3549 \hbox_overlap_right:n
3550 {
3551   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3552   {
3553     \skip_horizontal:N \l_@@_right_delim_dim
3554     \skip_horizontal:N \l_@@_right_margin_dim
3555     \skip_horizontal:N \l_@@_extra_right_margin_dim
3556     \@@_node_for_cell:
3557   }
3558 }
3559 \bool_gset_false:N \g_@@_empty_cell_bool
3560 }
3561 }
```

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

```
3562 \NewDocumentEnvironment { NiceArray } { }
3563 {
3564   \bool_gset_false:N \g_@@_delims_bool
3565   \str_if_empty:NT \g_@@_name_env_str
3566   { \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).

```
3567 \NiceArrayWithDelims . .
3568 }
3569 { \endNiceArrayWithDelims }
```

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

```
3570 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3571 {
3572   \NewDocumentEnvironment { #1 NiceArray } { }
```

```

3573 {
3574   \bool_gset_true:N \g_@@_delims_bool
3575   \str_if_empty:NT \g_@@_name_env_str
3576   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3577   \@@_test_if_math_mode:
3578   \NiceArrayWithDelims #2 #3
3579 }
3580 { \endNiceArrayWithDelims }
3581 }
3582 \@@_def_env:nnn p ( )
3583 \@@_def_env:nnn b [ ]
3584 \@@_def_env:nnn B \{ \}
3585 \@@_def_env:nnn v | |
3586 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3587 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3588 {
3589   \bool_set_false:N \l_@@_preamble_bool
3590   \tl_clear:N \l_tmpa_tl
3591   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3592   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3593   \tl_put_right:Nn \l_tmpa_tl
3594   {
3595     *
3596     {
3597       \int_case:nnF \l_@@_last_col_int
3598       {
3599         { -2 } { \c@MaxMatrixCols }
3600         { -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).

```

3601       }
3602       { \int_eval:n { \l_@@_last_col_int - 1 } }
3603     }
3604     { #2 }
3605   }
3606   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3607   \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3608 }
3609 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3610 \clist_map_inline:nn { p , b , B , v , V }
3611 {
3612   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3613   {
3614     \bool_gset_true:N \g_@@_delims_bool
3615     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3616     % added 2023/10/01
3617     \int_if_zero:nT \l_@@_last_col_int
3618     {
3619       \bool_set_true:N \l_@@_last_col_without_value_bool
3620       \int_set:Nn \l_@@_last_col_int { -1 }
3621     }
3622     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3623     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3624   }
3625   { \use:c { end #1 NiceArray } }
3626 }

```

We define also an environment `{NiceMatrix}`

```

3627 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3628 {
3629   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3630   % added 2023/10/01
3631   \int_if_zero:nT \l_@@_last_col_int
3632   {
3633     \bool_set_true:N \l_@@_last_col_without_value_bool
3634     \int_set:Nn \l_@@_last_col_int { -1 }
3635   }
3636   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3637   \bool_lazy_or:nnT
3638     { \clist_if_empty_p:N \l_@@_vlines_clist }
3639     { \l_@@_except_borders_bool }
3640     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3641   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3642 }
3643 { \endNiceArray }

```

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

```

3644 \cs_new_protected:Npn \@@_NotEmpty:
3645 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3646 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! O { } }
3647 {

```

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`.

```

3648   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3649     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3650   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3651   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3652   \tl_if_empty:NF \l_@@_short_caption_tl
3653   {
3654     \tl_if_empty:NT \l_@@_caption_tl
3655     {
3656       \@@_error_or_warning:n { short-caption~without~caption }
3657       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3658     }
3659   }
3660   \tl_if_empty:NF \l_@@_label_tl
3661   {
3662     \tl_if_empty:NT \l_@@_caption_tl
3663     { \@@_error_or_warning:n { label~without~caption } }
3664   }
3665   \NewDocumentEnvironment { TabularNote } { b }
3666   {
3667     \bool_if:NTF \l_@@_in_code_after_bool
3668       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3669       {
3670         \tl_if_empty:NF \g_@@_tabularnote_tl
3671         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3672         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3673       }
3674     }
3675   { }
3676   \bool_set_true:N \l_@@_tabular_bool
3677   \NiceArray { #2 }
3678 }
3679 { \endNiceArray }

```

```

3680 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3681 {
3682   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3683   \dim_zero_new:N \l_@@_width_dim
3684   \dim_set:Nn \l_@@_width_dim { #1 }
3685   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3686   \bool_set_true:N \l_@@_tabular_bool
3687   \NiceArray { #3 }
3688 }
3689 {
3690   \endNiceArray
3691   \int_if_zero:nT \g_@@_total_X_weight_int
3692     { \@@_error:n { NiceTabularX~without~X } }
3693 }

3694 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3695 {
3696   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3697   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3698   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3699   \bool_set_true:N \l_@@_tabular_bool
3700   \NiceArray { #3 }
3701 }
3702 { \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`).

```

3703 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3704 {
3705   \bool_lazy_all:nT
3706     {
3707       { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3708       \l_@@_hvlines_bool
3709       { ! \g_@@_delims_bool }
3710       { ! \l_@@_except_borders_bool }
3711     }
3712     {
3713       \bool_set_true:N \l_@@_except_borders_bool
3714       \clist_if_empty:NF \l_@@_corners_clist
3715         { \@@_error:n { hvlines,~rounded-corners-and-corners } }
3716       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3717         {
3718           \@@_stroke_block:nnn
3719             {
3720               rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3721               draw = \l_@@_rules_color_tl
3722             }
3723             { 1-1 }
3724             { \int_use:N \c@iRow - \int_use:N \c@jCol }
3725           }
3726     }
3727 }

3728 \cs_new_protected:Npn \@@_after_array:
3729 {
3730   \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.

```

3731 \bool_if:NT \g_@@_last_col_found_bool
3732 { \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`.

```

3733 \bool_if:NT \l_@@_last_col_without_value_bool
3734 { \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.

```

3735 \bool_if:NT \l_@@_last_row_without_value_bool
3736 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3737 \tl_build_gput_right:Nx \g_@@_aux_tl
3738 {
3739   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3740   {
3741     \int_use:N \l_@@_first_row_int ,
3742     \int_use:N \c@iRow ,
3743     \int_use:N \g_@@_row_total_int ,
3744     \int_use:N \l_@@_first_col_int ,
3745     \int_use:N \c@jCol ,
3746     \int_use:N \g_@@_col_total_int
3747   }
3748 }

```

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`).

```

3749 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3750 {
3751   \tl_build_gput_right:Nx \g_@@_aux_tl
3752   {
3753     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3754     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3755   }
3756 }
3757 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3758 {
3759   \tl_build_gput_right:Nx \g_@@_aux_tl
3760   {
3761     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3762     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3763     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3764     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3765   }
3766 }

```

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

```

3767 \@@_create_diag_nodes:

```

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

```

3768 \pgfpicture
3769 \int_step_inline:nn \c@iRow
3770 {
3771   \pgfnodelias
3772   { \@@_env: - ##1 - last }
3773   { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

3774     }
3775     \int_step_inline:nn \c@jCol
3776     {
3777         \pgfnodealias
3778         { \l_@@_env: - last - ##1 }
3779         { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3780     }
3781     \str_if_empty:NF \l_@@_name_str
3782     {
3783         \int_step_inline:nn \c@iRow
3784         {
3785             \pgfnodealias
3786             { \l_@@_name_str - ##1 - last }
3787             { \l_@@_env: - ##1 - \int_use:N \c@jCol }
3788         }
3789         \int_step_inline:nn \c@jCol
3790         {
3791             \pgfnodealias
3792             { \l_@@_name_str - last - ##1 }
3793             { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3794         }
3795     }
3796     \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.

```

3797     \bool_if:NT \l_@@_parallelize_diags_bool
3798     {
3799         \int_gzero_new:N \g_@@_ddots_int
3800         \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.

```

3801         \dim_gzero_new:N \g_@@_delta_x_one_dim
3802         \dim_gzero_new:N \g_@@_delta_y_one_dim
3803         \dim_gzero_new:N \g_@@_delta_x_two_dim
3804         \dim_gzero_new:N \g_@@_delta_y_two_dim
3805     }
3806     \int_zero_new:N \l_@@_initial_i_int
3807     \int_zero_new:N \l_@@_initial_j_int
3808     \int_zero_new:N \l_@@_final_i_int
3809     \int_zero_new:N \l_@@_final_j_int
3810     \bool_set_false:N \l_@@_initial_open_bool
3811     \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.

```

3812     \bool_if:NT \l_@@_small_bool
3813     {
3814         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3815         \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.

```

3816         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3817         { 0.6 \l_@@_xdots_shorten_start_dim }
3818         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

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

```

3819         { 0.6 \l_@@_xdots_shorten_end_dim }
3820     }

```

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

```

3821 \@@_draw_dotted_lines:

```

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

```

3822 \@@_compute_corners:

```

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

```

3823 \@@_adjust_pos_of_blocks_seq:
3824 \@@_deal_with_rounded_corners:
3825 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3826 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3827 \IfPackageLoadedTF { tikz }
3828 {
3829     \tikzset
3830     {
3831         every-picture / .style =
3832         {
3833             overlay ,
3834             remember~picture ,
3835             name~prefix = \@@_env: -
3836         }
3837     }
3838 }
3839 { }
3840 \cs_set_eq:NN \ialign \@@_old_ialign:
3841 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3842 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3843 \cs_set_eq:NN \OverBrace \@@_OverBrace
3844 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3845 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3846 \cs_set_eq:NN \line \@@_line
3847 \g_@@_pre_code_after_tl
3848 \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.

```

3849 \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).

```

3850 \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`).

```

3851 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3852 { \@@_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:`.

```

3853 \bool_set_true:N \l_@@_in_code_after_bool
3854 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3855 \scan_stop:
3856 \tl_gclear:N \g_nicematrix_code_after_tl
3857 \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.

```

3858 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3859 \tl_if_empty:NF \g_@@_pre_code_before_tl
3860 {
3861   \tl_build_gput_right:Nx \g_@@_aux_tl
3862   {
3863     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3864     { \exp_not:V \g_@@_pre_code_before_tl }
3865   }
3866   \tl_gclear:N \g_@@_pre_code_before_tl
3867 }
3868 \tl_if_empty:NF \g_nicematrix_code_before_tl
3869 {
3870   \tl_build_gput_right:Nx \g_@@_aux_tl
3871   {
3872     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3873     { \exp_not:V \g_nicematrix_code_before_tl }
3874   }
3875   \tl_gclear:N \g_nicematrix_code_before_tl
3876 }

3877 \str_gclear:N \g_@@_name_env_str
3878 \@@_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`.

```

3879 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3880 }

```

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`.

```

3881 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3882 { \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).

```

3883 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3884 {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`


```

3885 \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3886 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3887 }

```

The following command must *not* be protected.

```

3888 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3889 {
3890   { #1 }
3891   { #2 }
3892   {
3893     \int_compare:nNnTF { #3 } > { 99 }
3894     { \int_use:N \c@iRow }
3895     { #3 }
3896   }
3897   {
3898     \int_compare:nNnTF { #4 } > { 99 }
3899     { \int_use:N \c@jCol }
3900     { #4 }
3901   }
3902   { #5 }
3903 }

```

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).

```

3904 \hook_gput_code:nnn { begindocument } { . }
3905 {
3906   \cs_new_protected:Npx \@@_draw_dotted_lines:
3907   {
3908     \c_@@_pgfortikzpicture_tl
3909     \@@_draw_dotted_lines_i:
3910     \c_@@_endpgfortikzpicture_tl
3911   }
3912 }

```

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

```

3913 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3914 {
3915   \pgfrememberpicturerepositiononpagetrue
3916   \pgf@relevantforpicturesizefalse
3917   \g_@@_HVdotsfor_lines_tl
3918   \g_@@_Vdots_lines_tl
3919   \g_@@_Ddots_lines_tl
3920   \g_@@_Iddots_lines_tl
3921   \g_@@_Cdots_lines_tl
3922   \g_@@_Ldots_lines_tl
3923 }

3924 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3925 {
3926   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3927   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3928 }

```

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

```

3929 \pgfdeclareshape { @@_diag_node }
3930 {
3931   \savedanchor { \five }
3932   {
3933     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

3934     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3935   }
3936   \anchor { 5 } { \five }
3937   \anchor { center } { \pgfpointorigin }
3938 }

```

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).

```

3939 \cs_new_protected:Npn \@@_create_diag_nodes:
3940 {
3941   \pgfpicture
3942   \pgfrememberpicturepositiononpagetrue
3943   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3944   {
3945     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3946     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3947     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3948     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3949     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3950     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3951     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3952     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3953     \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@@_diag_node`) that we will construct.

```

3954     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3955     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3956     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3957     \str_if_empty:NF \l_@@_name_str
3958     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3959   }

```

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

```

3960   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3961   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3962   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3963   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3964   \pgfcoordinate
3965   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3966   \pgfnodealias
3967   { \@@_env: - last }
3968   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3969   \str_if_empty:NF \l_@@_name_str
3970   {
3971     \pgfnodealias
3972     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3973     { \@@_env: - \int_use:N \l_tmpa_int }
3974     \pgfnodealias
3975     { \l_@@_name_str - last }
3976     { \@@_env: - last }
3977   }
3978   \endpgfpicture
3979 }

```

17 We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on

its right.

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

The command `\l_@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
3980 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3981 {
```

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

```
3982 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3983 \int_set:Nn \l_@@_initial_i_int { #1 }
3984 \int_set:Nn \l_@@_initial_j_int { #2 }
3985 \int_set:Nn \l_@@_final_i_int { #1 }
3986 \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.

```
3987 \bool_set_false:N \l_@@_stop_loop_bool
3988 \bool_do_until:Nn \l_@@_stop_loop_bool
3989 {
3990 \int_add:Nn \l_@@_final_i_int { #3 }
3991 \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3992 \bool_set_false:N \l_@@_final_open_bool
3993 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3994 {
3995 \int_compare:nNnTF { #3 } = 1
3996 { \bool_set_true:N \l_@@_final_open_bool }
3997 {
3998 \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3999 { \bool_set_true:N \l_@@_final_open_bool }
4000 }
4001 }
4002 {
4003 \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
4004 {
4005 \int_compare:nNnTF { #4 } = { -1 }
4006 { \bool_set_true:N \l_@@_final_open_bool }
4007 }
4008 {
```

```

4009         \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4010         {
4011             \int_compare:nNnT { #4 } = 1
4012             { \bool_set_true:N \l_@@_final_open_bool }
4013         }
4014     }
4015 }
4016 \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.

```

4017 {

```

We do a step backwards.

```

4018     \int_sub:Nn \l_@@_final_i_int { #3 }
4019     \int_sub:Nn \l_@@_final_j_int { #4 }
4020     \bool_set_true:N \l_@@_stop_loop_bool
4021 }

```

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`.

```

4022 {
4023     \cs_if_exist:cTF
4024     {
4025         @@ _ dotted _
4026         \int_use:N \l_@@_final_i_int -
4027         \int_use:N \l_@@_final_j_int
4028     }
4029     {
4030         \int_sub:Nn \l_@@_final_i_int { #3 }
4031         \int_sub:Nn \l_@@_final_j_int { #4 }
4032         \bool_set_true:N \l_@@_final_open_bool
4033         \bool_set_true:N \l_@@_stop_loop_bool
4034     }
4035     {
4036         \cs_if_exist:cTF
4037         {
4038             pgf @ sh @ ns @ \@@_env:
4039             - \int_use:N \l_@@_final_i_int
4040             - \int_use:N \l_@@_final_j_int
4041         }
4042         { \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.

```

4043     {
4044         \cs_set:cpn
4045         {
4046             @@ _ dotted _
4047             \int_use:N \l_@@_final_i_int -
4048             \int_use:N \l_@@_final_j_int
4049         }
4050         { }
4051     }
4052 }
4053 }
4054 }

```

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

```

4055     \bool_set_false:N \l_@@_stop_loop_bool

```

```

4056 \bool_do_until:Nn \l_@@_stop_loop_bool
4057 {
4058   \int_sub:Nn \l_@@_initial_i_int { #3 }
4059   \int_sub:Nn \l_@@_initial_j_int { #4 }
4060   \bool_set_false:N \l_@@_initial_open_bool
4061   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4062   {
4063     \int_compare:nNnTF { #3 } = 1
4064     { \bool_set_true:N \l_@@_initial_open_bool }
4065     {
4066       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4067       { \bool_set_true:N \l_@@_initial_open_bool }
4068     }
4069   }
4070   {
4071     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4072     {
4073       \int_compare:nNnT { #4 } = 1
4074       { \bool_set_true:N \l_@@_initial_open_bool }
4075     }
4076     {
4077       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4078       {
4079         \int_compare:nNnT { #4 } = { -1 }
4080         { \bool_set_true:N \l_@@_initial_open_bool }
4081       }
4082     }
4083   }
4084   \bool_if:NTF \l_@@_initial_open_bool
4085   {
4086     \int_add:Nn \l_@@_initial_i_int { #3 }
4087     \int_add:Nn \l_@@_initial_j_int { #4 }
4088     \bool_set_true:N \l_@@_stop_loop_bool
4089   }
4090   {
4091     \cs_if_exist:cTF
4092     {
4093       @@ _ dotted _
4094       \int_use:N \l_@@_initial_i_int -
4095       \int_use:N \l_@@_initial_j_int
4096     }
4097     {
4098       \int_add:Nn \l_@@_initial_i_int { #3 }
4099       \int_add:Nn \l_@@_initial_j_int { #4 }
4100       \bool_set_true:N \l_@@_initial_open_bool
4101       \bool_set_true:N \l_@@_stop_loop_bool
4102     }
4103     {
4104       \cs_if_exist:cTF
4105       {
4106         pgf @ sh @ ns @ \@@_env:
4107         - \int_use:N \l_@@_initial_i_int
4108         - \int_use:N \l_@@_initial_j_int
4109       }
4110       { \bool_set_true:N \l_@@_stop_loop_bool }
4111       {
4112         \cs_set:cpn
4113         {
4114           @@ _ dotted _
4115           \int_use:N \l_@@_initial_i_int -
4116           \int_use:N \l_@@_initial_j_int
4117         }
4118         { }

```

```

4119     }
4120   }
4121 }
4122 }

```

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.

```

4123 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4124 {
4125   { \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`.

```

4126   { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4127   { \int_use:N \l_@@_final_i_int }
4128   { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4129   { } % for the name of the block
4130 }
4131 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analyse of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4132 \cs_new_protected:Npn \@@_open_shorten:
4133 {
4134   \bool_if:NT \l_@@_initial_open_bool
4135   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4136   \bool_if:NT \l_@@_final_open_bool
4137   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4138 }

```

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).

```

4139 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4140 {
4141   \int_set:Nn \l_@@_row_min_int 1
4142   \int_set:Nn \l_@@_col_min_int 1
4143   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4144   \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`.

```

4145 \seq_map_inline:Nn \g_@@_submatrix_seq
4146 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4147 }

```

#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.

```

4148 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4149 {
4150   \bool_if:nT
4151   {
4152     \int_compare_p:n { #3 <= #1 }
4153     && \int_compare_p:n { #1 <= #5 }
4154     && \int_compare_p:n { #4 <= #2 }
4155     && \int_compare_p:n { #2 <= #6 }
4156   }

```

```

4157     {
4158         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4159         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4160         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4161         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4162     }
4163 }

4164 \cs_new_protected:Npn \@@_set_initial_coords:
4165 {
4166     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4167     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4168 }
4169 \cs_new_protected:Npn \@@_set_final_coords:
4170 {
4171     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4172     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4173 }
4174 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4175 {
4176     \pgfpointanchor
4177     {
4178         \@@_env:
4179         - \int_use:N \l_@@_initial_i_int
4180         - \int_use:N \l_@@_initial_j_int
4181     }
4182     { #1 }
4183     \@@_set_initial_coords:
4184 }
4185 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4186 {
4187     \pgfpointanchor
4188     {
4189         \@@_env:
4190         - \int_use:N \l_@@_final_i_int
4191         - \int_use:N \l_@@_final_j_int
4192     }
4193     { #1 }
4194     \@@_set_final_coords:
4195 }
4196 \cs_new_protected:Npn \@@_open_x_initial_dim:
4197 {
4198     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4199     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4200     {
4201         \cs_if_exist:cT
4202         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4203         {
4204             \pgfpointanchor
4205             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4206             { west }
4207             \dim_set:Nn \l_@@_x_initial_dim
4208             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4209         }
4210     }

```

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

```

4211     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4212     {
4213         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4214         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4215         \dim_add:Nn \l_@@_x_initial_dim \col@sep
4216     }
4217 }

```

```

4218 \cs_new_protected:Npn \@@_open_x_final_dim:
4219 {
4220   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4221   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4222   {
4223     \cs_if_exist:cT
4224     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4225     {
4226       \pgfpointanchor
4227       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4228       { east }
4229       \dim_set:Nn \l_@@_x_final_dim
4230       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4231     }
4232   }

```

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

```

4233   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4234   {
4235     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4236     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4237     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4238   }
4239 }

```

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.

```

4240 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4241 {
4242   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4243   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4244   {
4245     \@@_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.

```

4246   \group_begin:
4247   \@@_open_shorten:
4248   \int_if_zero:nTF { #1 }
4249   { \color { nicematrix-first-row } }
4250   {

```

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.

```

4251       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4252       { \color { nicematrix-last-row } }
4253   }
4254   \keys_set:nn { NiceMatrix / xdots } { #3 }
4255   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4256   \@@_actually_draw_Ldots:
4257   \group_end:
4258 }
4259 }

```

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`.

```

4260 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4261 {
4262   \bool_if:NTF \l_@@_initial_open_bool
4263   {
4264     \@@_open_x_initial_dim:
4265     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4266     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4267   }
4268   { \@@_set_initial_coords_from_anchor:n { base-east } }
4269   \bool_if:NTF \l_@@_final_open_bool
4270   {
4271     \@@_open_x_final_dim:
4272     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4273     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4274   }
4275   { \@@_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.

```

4276   \bool_lazy_all:nTF
4277   {
4278     \l_@@_initial_open_bool
4279     \l_@@_final_open_bool
4280     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4281   }
4282   {
4283     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4284     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4285   }

```

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 (?).

```

4286   {
4287     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4288     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4289   }
4290   \@@_draw_line:
4291 }

```

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.

```

4292 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4293 {
4294   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4295   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4296   {
4297     \@@_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.

```

4298   \group_begin:
4299   \@@_open_shorten:
4300   \int_if_zero:nTF { #1 }
4301   { \color { nicematrix-first-row } }
4302   {

```

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.

```

4303         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4304         { \color { nicematrix-last-row } }
4305     }
4306     \keys_set:nn { NiceMatrix / xdots } { #3 }
4307     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4308     \@@_actually_draw_Cdots:
4309 \group_end:
4310 }
4311 }

```

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`.

```

4312 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4313 {
4314     \bool_if:NTF \l_@@_initial_open_bool
4315     { \@@_open_x_initial_dim: }
4316     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4317     \bool_if:NTF \l_@@_final_open_bool
4318     { \@@_open_x_final_dim: }
4319     { \@@_set_final_coords_from_anchor:n { mid-west } }
4320     \bool_lazy_and:nnTF
4321     \l_@@_initial_open_bool
4322     \l_@@_final_open_bool
4323     {
4324         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4325         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4326         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4327         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4328         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4329     }
4330     {
4331         \bool_if:NT \l_@@_initial_open_bool
4332         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4333         \bool_if:NT \l_@@_final_open_bool
4334         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4335     }
4336     \@@_draw_line:
4337 }
4338 \cs_new_protected:Npn \@@_open_y_initial_dim:
4339 {
4340     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4341     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4342     {
4343         \cs_if_exist:cT
4344         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4345         {
4346             \pgfpointanchor
4347             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4348             { north }
4349             \dim_set:Nn \l_@@_y_initial_dim

```

```

4350         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4351     }
4352 }
4353 % modified 2023-08-10
4354 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4355 {
4356     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4357     \dim_set:Nn \l_@@_y_initial_dim
4358     {
4359         \fp_to_dim:n
4360         {
4361             \pgf@y
4362             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4363         }
4364     }
4365 }
4366 }
4367 \cs_new_protected:Npn \@@_open_y_final_dim:
4368 {
4369     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4370     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4371     {
4372         \cs_if_exist:cT
4373         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4374         {
4375             \pgfpointanchor
4376             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4377             { south }
4378             \dim_set:Nn \l_@@_y_final_dim
4379             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4380         }
4381     }
4382 % modified 2023-08-10
4383 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4384 {
4385     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4386     \dim_set:Nn \l_@@_y_final_dim
4387     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4388 }
4389 }

```

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.

```

4390 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4391 {
4392     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4393     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4394     {
4395         \@@_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.

```

4396     \group_begin:
4397     \@@_open_shorten:
4398     \int_if_zero:nTF { #2 }
4399     { \color { nicematrix-first-col } }
4400     {
4401         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4402         { \color { nicematrix-last-col } }
4403     }
4404     \keys_set:nn { NiceMatrix / xdots } { #3 }
4405     \tl_if_empty:VF \l_@@_xdots_color_tl
4406     { \color { \l_@@_xdots_color_tl } }
4407     \@@_actually_draw_Vdots:

```

```

4408     \group_end:
4409   }
4410 }

```

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`.

```

4411 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4412 {

```

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

```

4413     \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.

```

4414     {
4415       \@@_open_y_initial_dim:
4416       \@@_open_y_final_dim:
4417       \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4418     {
4419       \@@_qpoint:n { col - 1 }
4420       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4421       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4422       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4423       % \bool_if:NT \g_@@_delims_bool
4424       % {
4425         \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4426       % }
4427     }
4428     {
4429       \bool_lazy_and:nnTF
4430       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4431       { \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”.

```

4432     {
4433       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4434       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4435       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4436       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4437       % \bool_if:NT \g_@@_delims_bool
4438       % {
4439         \dim_add:Nn
4440         \l_@@_x_initial_dim
4441         \c_@@_shift_exterior_Vdots_dim
4442       % }
4443     }

```

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

```

4444     {
4445       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4446       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4447       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }

```

```

4448         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4449     }
4450 }
4451 }

```

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.

```

4452 {
4453     \bool_set_false:N \l_tmpa_bool
4454     \bool_lazy_and:nnT
4455     { ! \l_@@_initial_open_bool }
4456     { ! \l_@@_final_open_bool }
4457     {
4458         \@@_set_initial_coords_from_anchor:n { south-west }
4459         \@@_set_final_coords_from_anchor:n { north-west }
4460         \bool_set:Nn \l_tmpa_bool
4461         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4462     }

```

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

```

4463     \bool_if:NTF \l_@@_initial_open_bool
4464     {
4465         \@@_open_y_initial_dim:
4466         \@@_set_final_coords_from_anchor:n { north }
4467         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4468     }
4469     {
4470         \@@_set_initial_coords_from_anchor:n { south }
4471         \bool_if:NTF \l_@@_final_open_bool
4472         \@@_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.

```

4473     {
4474         \@@_set_final_coords_from_anchor:n { north }
4475         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4476         {
4477             \dim_set:Nn \l_@@_x_initial_dim
4478             {
4479                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4480                 \l_@@_x_initial_dim \l_@@_x_final_dim
4481             }
4482         }
4483     }
4484 }
4485 }
4486 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4487 \@@_draw_line:
4488 }

```

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.

```

4489 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4490 {
4491     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4492     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4493     {
4494         \@@_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.

```

4495     \group_begin:
4496     \@@_open_shorten:
4497     \keys_set:nn { NiceMatrix / xdots } { #3 }
4498     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4499     \@@_actually_draw_Ddots:
4500     \group_end:
4501   }
4502 }

```

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.`

```

4503 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4504 {
4505   \bool_if:NTF \l_@@_initial_open_bool
4506   {
4507     \@@_open_y_initial_dim:
4508     \@@_open_x_initial_dim:
4509   }
4510   { \@@_set_initial_coords_from_anchor:n { south-east } }
4511   \bool_if:NTF \l_@@_final_open_bool
4512   {
4513     \@@_open_x_final_dim:
4514     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4515   }
4516   { \@@_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.

```

4517   \bool_if:NT \l_@@_parallelize_diags_bool
4518   {
4519     \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).

```

4520     \int_compare:nNnTF \g_@@_ddots_int = 1

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4521   {
4522     \dim_gset:Nn \g_@@_delta_x_one_dim
4523     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4524     \dim_gset:Nn \g_@@_delta_y_one_dim
4525     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4526   }

```

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`.

```

4527   {
4528     \dim_set:Nn \l_@@_y_final_dim
4529     {
4530       \l_@@_y_initial_dim +

```

```

4531         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4532         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4533     }
4534 }
4535 }
4536 \@@_draw_line:
4537 }

```

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.

```

4538 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4539 {
4540     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4541     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4542     {
4543         \@@_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.

```

4544     \group_begin:
4545     \@@_open_shorten:
4546     \keys_set:nn { NiceMatrix / xdots } { #3 }
4547     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4548     \@@_actually_draw_Iddots:
4549     \group_end:
4550 }
4551 }

```

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`.

```

4552 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4553 {
4554     \bool_if:NTF \l_@@_initial_open_bool
4555     {
4556         \@@_open_y_initial_dim:
4557         \@@_open_x_initial_dim:
4558     }
4559     { \@@_set_initial_coords_from_anchor:n { south~west } }
4560     \bool_if:NTF \l_@@_final_open_bool
4561     {
4562         \@@_open_y_final_dim:
4563         \@@_open_x_final_dim:
4564     }
4565     { \@@_set_final_coords_from_anchor:n { north~east } }
4566     \bool_if:NT \l_@@_parallelize_diags_bool
4567     {
4568         \int_gincr:N \g_@@_iddots_int
4569         \int_compare:nNnTF \g_@@_iddots_int = 1
4570         {
4571             \dim_gset:Nn \g_@@_delta_x_two_dim
4572             { \l_@@_x_final_dim - \l_@@_x_initial_dim }

```

```

4573         \dim_gset:Nn \g_@@_delta_y_two_dim
4574         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4575     }
4576     {
4577         \dim_set:Nn \l_@@_y_final_dim
4578         {
4579             \l_@@_y_initial_dim +
4580             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4581             \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4582         }
4583     }
4584 }
4585 \@@_draw_line:
4586 }

```

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`

```

4587 \cs_new_protected:Npn \@@_draw_line:
4588 {
4589     \pgfrememberpicturepositiononpagetrue
4590     \pgf@relevantforpicturesizefalse
4591     \bool_lazy_or:nnTF
4592     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4593     \l_@@_dotted_bool
4594     \@@_draw_standard_dotted_line:
4595     \@@_draw_unstandard_dotted_line:
4596 }

```

We have to do a special construction with `\exp_args:NV` to be able to put in the list of options in the correct place in the Tikz instruction.

```

4597 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4598 {
4599     \begin { scope }
4600     \@@_draw_unstandard_dotted_line:o
4601     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4602 }

```

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.

```

4603 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4604 {
4605     \@@_draw_unstandard_dotted_line:nVVV
4606     { #1 }

```



```

4607 \l_@@_xdots_up_tl
4608 \l_@@_xdots_down_tl
4609 \l_@@_xdots_middle_tl
4610 }
4611 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4612 \hook_gput_code:nnn { begindocument } { . }
4613 {
4614   \IfPackageLoadedTF { tikz }
4615   {
4616     \tikzset
4617     {
4618       @@_node_above / .style = { sloped , above } ,
4619       @@_node_below / .style = { sloped , below } ,
4620       @@_node_middle / .style =
4621       {
4622         sloped ,
4623         inner~sep = \c_@@_innersep_middle_dim
4624       }
4625     }
4626   }
4627   { }
4628 }

4629 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4630 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shortenu>` and `shortenu<` 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.

```

4631 \dim_zero_new:N \l_@@_l_dim
4632 \dim_set:Nn \l_@@_l_dim
4633 {
4634   \fp_to_dim:n
4635   {
4636     sqrt
4637     (
4638       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4639       +
4640       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4641     )
4642   }
4643 }
4644 \bool_lazy_and:nnT % security
4645 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } < \c_@@_max_l_dim }
4646 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > { 1 pt } }
4647 {
4648   \dim_set:Nn \l_tmpa_dim
4649   {
4650     \l_@@_x_initial_dim
4651     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4652     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4653   }
4654   \dim_set:Nn \l_tmpb_dim
4655   {
4656     \l_@@_y_initial_dim
4657     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4658     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4659     }
4660     \dim_set:Nn \l_@@_tmpc_dim
4661     {
4662         \l_@@_x_final_dim
4663         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4664         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4665     }
4666     \dim_set:Nn \l_@@_tmpd_dim
4667     {
4668         \l_@@_y_final_dim
4669         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4670         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4671     }
4672     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4673     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4674     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4675     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4676 }

```

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

```

4677     \bool_if:NT \l_@@_xdots_h_labels_bool
4678     {
4679         \tikzset
4680         {
4681             @@_node_above / .style = { auto = left } ,
4682             @@_node_below / .style = { auto = right } ,
4683             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4684         }
4685     }
4686     \tl_if_empty:nF { #4 }
4687     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4688     \draw
4689     [ #1 ]
4690     ( \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).

```

4691     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4692     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4693     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4694     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4695     \end { scope }
4696 }
4697 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n V V V }

```

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

```

4698 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4699 {
4700     \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.

```

4701     \dim_zero_new:N \l_@@_l_dim
4702     \dim_set:Nn \l_@@_l_dim
4703     {
4704         \fp_to_dim:n
4705         {
4706             sqrt
4707             (
4708                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4709                 +
4710                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4711             )

```

```

4712     }
4713 }

```

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.

```

4714     \bool_lazy_or:nnF
4715     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4716     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4717     \@@_draw_standard_dotted_line_i:
4718 \group_end:
4719 \bool_lazy_all:nF
4720 {
4721   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4722   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4723   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4724 }
4725 \l_@@_labels_standard_dotted_line:
4726 }
4727 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4728 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4729 {

```

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

```

4730   \int_set:Nn \l_tmpa_int
4731   {
4732     \dim_ratio:nn
4733     {
4734       \l_@@_l_dim
4735       - \l_@@_xdots_shorten_start_dim
4736       - \l_@@_xdots_shorten_end_dim
4737     }
4738     \l_@@_xdots_inter_dim
4739   }

```

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

```

4740   \dim_set:Nn \l_tmpa_dim
4741   {
4742     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4743     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4744   }
4745   \dim_set:Nn \l_tmpb_dim
4746   {
4747     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4748     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4749   }

```

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.

```

4750   \dim_gadd:Nn \l_@@_x_initial_dim
4751   {
4752     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4753     \dim_ratio:nn
4754     {
4755       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4756       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4757     }
4758     { 2 \l_@@_l_dim }
4759   }
4760   \dim_gadd:Nn \l_@@_y_initial_dim

```

```

4761 {
4762   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4763   \dim_ratio:nn
4764   {
4765     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4766     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4767   }
4768   { 2 \l_@@_l_dim }
4769 }
4770 \pgf@relevantforpicturesizefalse
4771 \int_step_inline:nnn 0 \l_tmpa_int
4772 {
4773   \pgfpathcircle
4774   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4775   { \l_@@_xdots_radius_dim }
4776   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4777   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4778 }
4779 \pgfusepathqfill
4780 }

4781 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4782 {
4783   \pgfscope
4784   \pgftransformshift
4785   {
4786     \pgfpointlineattime { 0.5 }
4787     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4788     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4789   }
4790   \fp_set:Nn \l_tmpa_fp
4791   {
4792     atand
4793     (
4794       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4795       \l_@@_x_final_dim - \l_@@_x_initial_dim
4796     )
4797   }
4798   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4799   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4800   \tl_if_empty:NF \l_@@_xdots_middle_tl
4801   {
4802     \begin { pgfscope }
4803     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4804     \pgfnode
4805     { rectangle }
4806     { center }
4807     {
4808       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4809       {
4810         \c_math_toggle_token
4811         \scriptstyle \l_@@_xdots_middle_tl
4812         \c_math_toggle_token
4813       }
4814     }
4815     { }
4816     {
4817       \pgfsetfillcolor { white }
4818       \pgfusepath { fill }
4819     }
4820     \end { pgfscope }
4821   }
4822   \tl_if_empty:NF \l_@@_xdots_up_tl

```

```

4823 {
4824   \pgfnode
4825   { rectangle }
4826   { south }
4827   {
4828     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4829     {
4830       \c_math_toggle_token
4831       \scriptstyle \l_@@_xdots_up_tl
4832       \c_math_toggle_token
4833     }
4834   }
4835   { }
4836   { \pgfusepath { } }
4837 }
4838 \tl_if_empty:NF \l_@@_xdots_down_tl
4839 {
4840   \pgfnode
4841   { rectangle }
4842   { north }
4843   {
4844     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4845     {
4846       \c_math_toggle_token
4847       \scriptstyle \l_@@_xdots_down_tl
4848       \c_math_toggle_token
4849     }
4850   }
4851   { }
4852   { \pgfusepath { } }
4853 }
4854 \endpgfscope
4855 }

```

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{_\begin{document}}{__}` and the *arg spec* will be rescanned.

```

4856 \hook_gput_code:nnn { \begin{document} } { . }
4857 {
4858   \tl_set:Nn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4859   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4860   \cs_new_protected:Npn \@@_Ldots
4861     { \@@_collect_options:n { \@@_Ldots_i } }
4862   \exp_args:NNV \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4863     {
4864       \int_if_zero:nTF \c@jCol
4865       { \@@_error:nn { in~first~col } \Ldots }
4866       {
4867         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4868         { \@@_error:nn { in~last~col } \Ldots }
4869         {

```

```

4870         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4871         { #1 , down = #2 , up = #3 , middle = #4 }
4872     }
4873 }
4874 \bool_if:NF \l_@@_nullify_dots_bool
4875 { \phantom { \ensuremath { \@@_old_ldots } } }
4876 \bool_gset_true:N \g_@@_empty_cell_bool
4877 }

4878 \cs_new_protected:Npn \@@_Cdots
4879 { \@@_collect_options:n { \@@_Cdots_i } }
4880 \exp_args:NNV \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4881 {
4882     \int_if_zero:nTF \c@jCol
4883     { \@@_error:nn { in~first~col } \Cdots }
4884     {
4885         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4886         { \@@_error:nn { in~last~col } \Cdots }
4887         {
4888             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4889             { #1 , down = #2 , up = #3 , middle = #4 }
4890         }
4891     }
4892     \bool_if:NF \l_@@_nullify_dots_bool
4893     { \phantom { \ensuremath { \@@_old_cdots } } }
4894     \bool_gset_true:N \g_@@_empty_cell_bool
4895 }

4896 \cs_new_protected:Npn \@@_Vdots
4897 { \@@_collect_options:n { \@@_Vdots_i } }
4898 \exp_args:NNV \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4899 {
4900     \int_if_zero:nTF \c@iRow
4901     { \@@_error:nn { in~first~row } \Vdots }
4902     {
4903         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4904         { \@@_error:nn { in~last~row } \Vdots }
4905         {
4906             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4907             { #1 , down = #2 , up = #3 , middle = #4 }
4908         }
4909     }
4910     \bool_if:NF \l_@@_nullify_dots_bool
4911     { \phantom { \ensuremath { \@@_old_vdots } } }
4912     \bool_gset_true:N \g_@@_empty_cell_bool
4913 }

4914 \cs_new_protected:Npn \@@_Ddots
4915 { \@@_collect_options:n { \@@_Ddots_i } }
4916 \exp_args:NNV \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4917 {
4918     \int_case:nnF \c@iRow
4919     {
4920         0 { \@@_error:nn { in~first~row } \Ddots }
4921         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4922     }
4923     {
4924         \int_case:nnF \c@jCol
4925         {
4926             0 { \@@_error:nn { in~first~col } \Ddots }
4927             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }

```

```

4928     }
4929     {
4930         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4931         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4932         { #1 , down = #2 , up = #3 , middle = #4 }
4933     }
4934
4935     }
4936     \bool_if:NF \l_@@_nullify_dots_bool
4937     { \phantom { \ensuremath { \@@_old_ddots } } }
4938     \bool_gset_true:N \g_@@_empty_cell_bool
4939 }

```

```

4940 \cs_new_protected:Npn \@@_Iddots
4941 { \@@_collect_options:n { \@@_Iddots_i } }
4942 \exp_args:NNV \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4943 {
4944     \int_case:nnF \c@iRow
4945     {
4946         0 { \@@_error:nn { in-first~row } \Iddots }
4947         \l_@@_last_row_int { \@@_error:nn { in-last~row } \Iddots }
4948     }
4949     {
4950         \int_case:nnF \c@jCol
4951         {
4952             0 { \@@_error:nn { in-first~col } \Iddots }
4953             \l_@@_last_col_int { \@@_error:nn { in-last~col } \Iddots }
4954         }
4955         {
4956             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4957             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4958             { #1 , down = #2 , up = #3 , middle = #4 }
4959         }
4960     }
4961     \bool_if:NF \l_@@_nullify_dots_bool
4962     { \phantom { \ensuremath { \@@_old_iddots } } }
4963     \bool_gset_true:N \g_@@_empty_cell_bool
4964 }
4965 }

```

End of the \AddToHook.

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

```

4966 \keys_define:nn { NiceMatrix / Ddots }
4967 {
4968     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4969     draw-first .default:n = true ,
4970     draw-first .value_forbidden:n = true
4971 }

```

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

```

4972 \cs_new_protected:Npn \@@_Hspace:
4973 {
4974     \bool_gset_true:N \g_@@_empty_cell_bool
4975     \hspace
4976 }

```

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.

```

4977 \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`.

```

4978 \cs_new:Npn \@@_Hdotsfor:
4979 {
4980   \bool_lazy_and:nnTF
4981     { \int_if_zero_p:n \c@jCol }
4982     { \int_if_zero_p:n \l_@@_first_col_int }
4983     {
4984       \bool_if:NTF \g_@@_after_col_zero_bool
4985       {
4986         \multicolumn { 1 } { c } { }
4987         \@@_Hdotsfor_i
4988       }
4989       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4990     }
4991   {
4992     \multicolumn { 1 } { c } { }
4993     \@@_Hdotsfor_i
4994   }
4995 }

```

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:`).

```

4996 \hook_gput_code:nnn { begindocument } { . }
4997 {
4998   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4999   \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.

```

5000 \cs_new_protected:Npn \@@_Hdotsfor_i
5001 { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5002 \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5003 {
5004   \tl_gput_right:Nx \g_@@_Hvdotsfor_lines_tl
5005   {
5006     \@@_Hdotsfor:nnnn
5007     { \int_use:N \c@iRow }
5008     { \int_use:N \c@jCol }
5009     { #2 }
5010     {
5011       #1 , #3 ,
5012       down = \exp_not:n { #4 } ,
5013       up = \exp_not:n { #5 } ,
5014       middle = \exp_not:n { #6 }
5015     }
5016   }
5017   \prg_replicate:nn { #2 - 1 }
5018   {
5019     &
5020     \multicolumn { 1 } { c } { }
5021     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i: % added 2023-08-26
5022   }
5023 }
5024 }

```

```

5025 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5026 {
5027   \bool_set_false:N \l_@@_initial_open_bool
5028   \bool_set_false:N \l_@@_final_open_bool

```


For the row, it's easy.

```
5029 \int_set:Nn \l_@@_initial_i_int { #1 }
5030 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5031 \int_compare:nNnTF { #2 } = 1
5032 {
5033   \int_set:Nn \l_@@_initial_j_int 1
5034   \bool_set_true:N \l_@@_initial_open_bool
5035 }
5036 {
5037   \cs_if_exist:cTF
5038   {
5039     pgf @ sh @ ns @ \@@_env:
5040     - \int_use:N \l_@@_initial_i_int
5041     - \int_eval:n { #2 - 1 }
5042   }
5043   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5044   {
5045     \int_set:Nn \l_@@_initial_j_int { #2 }
5046     \bool_set_true:N \l_@@_initial_open_bool
5047   }
5048 }
5049 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5050 {
5051   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5052   \bool_set_true:N \l_@@_final_open_bool
5053 }
5054 {
5055   \cs_if_exist:cTF
5056   {
5057     pgf @ sh @ ns @ \@@_env:
5058     - \int_use:N \l_@@_final_i_int
5059     - \int_eval:n { #2 + #3 }
5060   }
5061   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5062   {
5063     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5064     \bool_set_true:N \l_@@_final_open_bool
5065   }
5066 }
5067 \group_begin:
5068 \@@_open_shorten:
5069 \int_if_zero:nTF { #1 }
5070 { \color { nicematrix-first-row } }
5071 {
5072   \int_compare:nNnT { #1 } = \g_@@_row_total_int
5073   { \color { nicematrix-last-row } }
5074 }
5075
5076 \keys_set:nn { NiceMatrix / xdots } { #4 }
5077 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5078 \@@_actually_draw_ldots:
5079 \group_end:
```

We declare all the cells concerned by the `\Hdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```
5080 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5081 { \cs_set:cpn { @@ _ dotted _ #1 - #1 } { } }
5082 }
```

```

5083 \hook_gput_code:nnn { begindocument } { . }
5084 {
5085   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5086   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5087   \cs_new_protected:Npn \@@_Vdotsfor:
5088     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5089   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5090     {
5091       \bool_gset_true:N \g_@@_empty_cell_bool
5092       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5093         {
5094           \@@_Vdotsfor:nnnn
5095             { \int_use:N \c@iRow }
5096             { \int_use:N \c@jCol }
5097             { #2 }
5098             {
5099               #1 , #3 ,
5100               down = \exp_not:n { #4 } ,
5101               up = \exp_not:n { #5 } ,
5102               middle = \exp_not:n { #6 }
5103             }
5104           }
5105         }
5106     }

5107 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5108 {
5109   \bool_set_false:N \l_@@_initial_open_bool
5110   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5111   \int_set:Nn \l_@@_initial_j_int { #2 }
5112   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5113   \int_compare:nNnTF { #1 } = 1
5114   {
5115     \int_set:Nn \l_@@_initial_i_int 1
5116     \bool_set_true:N \l_@@_initial_open_bool
5117   }
5118   {
5119     \cs_if_exist:cTF
5120     {
5121       pgf @ sh @ ns @ \@@_env:
5122       - \int_eval:n { #1 - 1 }
5123       - \int_use:N \l_@@_initial_j_int
5124     }
5125     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5126     {
5127       \int_set:Nn \l_@@_initial_i_int { #1 }
5128       \bool_set_true:N \l_@@_initial_open_bool
5129     }
5130   }
5131   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5132   {
5133     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5134     \bool_set_true:N \l_@@_final_open_bool
5135   }
5136   {
5137     \cs_if_exist:cTF
5138     {
5139       pgf @ sh @ ns @ \@@_env:
5140       - \int_eval:n { #1 + #3 }

```

```

5141         - \int_use:N \l_@@_final_j_int
5142     }
5143     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5144     {
5145         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5146         \bool_set_true:N \l_@@_final_open_bool
5147     }
5148 }
5149 \group_begin:
5150 \@@_open_shorten:
5151 \int_if_zero:nTF { #2 }
5152 { \color { nicematrix-first-col } }
5153 {
5154     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5155     { \color { nicematrix-last-col } }
5156 }
5157 \keys_set:nn { NiceMatrix / xdots } { #4 }
5158 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5159 \@@_actually_draw_Vdots:
5160 \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).

```

5161     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5162     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5163 }

```

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

```

5164 \NewDocumentCommand \@@_rotate: { 0 { } }
5165 {
5166     \peek_remove_spaces:n
5167     {
5168         \bool_gset_true:N \g_@@_rotate_bool
5169         \keys_set:nn { NiceMatrix / rotate } { #1 }
5170     }
5171 }
5172 \keys_define:nn { NiceMatrix / rotate }
5173 {
5174     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5175     c .value_forbidden:n = true ,
5176     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5177 }

```

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

```

5178 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5179 {
5180   \tl_if_empty:nTF { #2 }
5181     { #1 }
5182     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5183 }
5184 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5185 { \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).

```

5186 \hook_gput_code:nnn { begindocument } { . }
5187 {
5188   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5189   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5190   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
5191     {
5192       \group_begin:
5193       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5194       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5195       \use:e
5196       {
5197         \@@_line_i:nn
5198         { \@@_double_int_eval:n #2 - \q_stop }
5199         { \@@_double_int_eval:n #3 - \q_stop }
5200       }
5201       \group_end:
5202     }
5203 }
5204 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5205 {
5206   \bool_set_false:N \l_@@_initial_open_bool
5207   \bool_set_false:N \l_@@_final_open_bool
5208   \bool_if:nTF
5209     {
5210       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
5211       ||
5212       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
5213     }
5214     {
5215       \@@_error:nnn { unknown-cell-for~line~in~CodeAfter } { #1 } { #2 }
5216     }
5217 }
```

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

```

5217   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5218 }
5219 \hook_gput_code:nnn { begindocument } { . }
5220 {
5221   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5222   {
```

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:`.

```

5223   \c_@@_pgfortikzpicture_tl
```

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

```

5224     \@@_draw_line_iii:nn { #1 } { #2 }
5225     \c_@@_endpgfortikzpicture_tl
5226   }
5227 }

```

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

```

5228 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5229 {
5230   \pgfrememberpicturepositiononpagetrue
5231   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5232   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5233   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5234   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5235   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5236   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5237   \@@_draw_line:
5238 }

```

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_then: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_then:nn is *not* protected.

#1 is the first row *after* the scope of the instructions in **#2**

```

5239 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5240 { \int_compare:nNnT { \int_use:N \c@iRow } < { #1 } { #2 } }

```

\@@_put_in_row_style will be used several times by \RowStyle.

```

5241 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5242 {
5243   \tl_gput_right:Nx \g_@@_row_style_tl
5244   {

```

Be careful, \exp_not:N\@@_if_row_less_than:nn can't be replaced by a protected version of \@@_if_row_less_than:nn.

```

5245     \exp_not:N
5246     \@@_if_row_less_than:nn
5247     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5248     { \exp_not:n { #1 } }
5249   }
5250 }
5251 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5252 \keys_define:nn { NiceMatrix / RowStyle }
5253 {
5254   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5255   cell-space-top-limit .initial:n = \c_zero_dim ,
5256   cell-space-top-limit .value_required:n = true ,

```

```

5257 cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5258 cell-space-bottom-limit .initial:n = \c_zero_dim ,
5259 cell-space-bottom-limit .value_required:n = true ,
5260 cell-space-limits .meta:n =
5261 {
5262   cell-space-top-limit = #1 ,
5263   cell-space-bottom-limit = #1 ,
5264 } ,
5265 color .tl_set:N = \l_@@_color_tl ,
5266 color .value_required:n = true ,
5267 bold .bool_set:N = \l_tmpa_bool ,
5268 bold .default:n = true ,
5269 bold .initial:n = false ,
5270 nb-rows .code:n =
5271   \str_if_eq:nnTF { #1 } { * }
5272   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5273   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5274 nb-rows .value_required:n = true ,
5275 rowcolor .tl_set:N = \l_tmpa_tl ,
5276 rowcolor .value_required:n = true ,
5277 rowcolor .initial:n = ,
5278 unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5279 }

```

```

5280 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5281 {
5282   \group_begin:
5283   \tl_clear:N \l_tmpa_tl % value of \rowcolor
5284   \tl_clear:N \l_@@_color_tl
5285   \int_set:Nn \l_@@_key_nb_rows_int 1
5286   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5287   \tl_if_empty:NF \l_tmpa_tl
5288   {

```

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

```

5289     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5290     {

```

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

```

5291       \@@_exp_color_arg:N \@@_rectanglecolor \l_tmpa_tl
5292       { \int_use:N \c_iRow - \int_use:N \c_jCol }
5293       { \int_use:N \c_iRow - * }
5294     }

```

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

```

5295     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
5296     {
5297       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5298       {
5299         \@@_exp_color_arg:N \@@_rowcolor \l_tmpa_tl
5300         {
5301           \int_eval:n { \c_iRow + 1 }
5302           - \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5303         }
5304       }
5305     }
5306   }
5307   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5308   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5309   {

```

```

5310 \@@_put_in_row_style:n
5311 {
5312   \exp_not:n
5313   {
5314     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
5315     {
5316       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5317       { \dim_use:N \l_tmpa_dim }
5318     }
5319   }
5320 }
5321 }

```

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

```

5322 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5323 {
5324   \@@_put_in_row_style:n
5325   {
5326     \exp_not:n
5327     {
5328       \tl_gput_right:Nn \g_@@_cell_after_hook_tl
5329       {
5330         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5331         { \dim_use:N \l_tmpb_dim }
5332       }
5333     }
5334   }
5335 }

```

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

```

5336 \tl_if_empty:NF \l_@@_color_tl
5337 {
5338   \@@_put_in_row_style:e
5339   {
5340     \mode_leave_vertical:
5341     \@@_color:n { \l_@@_color_tl }
5342   }
5343 }

```

`\l_tmpa_bool` is the value of the key `bold`.

```

5344 \bool_if:NT \l_tmpa_bool
5345 {
5346   \@@_put_in_row_style:n
5347   {
5348     \exp_not:n
5349     {
5350       \if_mode_math:
5351         \c_math_toggle_token
5352         \bfseries \boldmath
5353         \c_math_toggle_token
5354       \else:
5355         \bfseries \boldmath
5356       \fi:
5357     }
5358   }
5359 }
5360 \group_end:
5361 \g_@@_row_style_tl
5362 \ignorespaces
5363 }

```

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).

```
5364 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5365 {
```

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.

```
5366 \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`.

```
5367 \str_if_in:nnF { #1 } { !! }
5368 {
5369 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5370 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5371 }
5372 \int_if_zero:nTF \l_tmpa_int
```

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

```
5373 {
5374 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5375 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5376 }
```

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

```
5377 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5378 }

5379 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5380 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5381 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5382 {
5383 \dim_compare:nnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5384 {
```


The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5385     \group_begin:
5386     \pgfsetcornersarced
5387     {
5388         \pgfpoint
5389         { \l_@@_tab_rounded_corners_dim }
5390         { \l_@@_tab_rounded_corners_dim }
5391     }

```

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.

```

5392     \bool_if:NTF \l_@@_hvlines_bool
5393     {
5394         \pgfpathrectanglecorners
5395         {
5396             \pgfpointadd
5397             { \@@_qpoint:n { row-1 } }
5398             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5399         }
5400         {
5401             \pgfpointadd
5402             {
5403                 \@@_qpoint:n
5404                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5405             }
5406             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5407         }
5408     }
5409     {
5410         \pgfpathrectanglecorners
5411         { \@@_qpoint:n { row-1 } }
5412         {
5413             \pgfpointadd
5414             {
5415                 \@@_qpoint:n
5416                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5417             }
5418             { \pgfpoint \c_zero_dim \arrayrulewidth }
5419         }
5420     }
5421     \pgfusepath { clip }
5422     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5423     }
5424 }

```

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`).

```

5425 \cs_new_protected:Npn \@@_actually_color:
5426 {
5427     \pgfpicture
5428     \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.

```

5429     \@@_clip_with_rounded_corners:
5430     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5431     {
5432         \int_compare:nNnTF { ##1 } = 1

```

```

5433     {
5434         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5435         \use:c { g_@@_color _ 1 _tl }
5436         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5437     }
5438     {
5439         \begin { pgfscope }
5440             \@@_color_opacity ##2
5441             \use:c { g_@@_color _ ##1 _tl }
5442             \tl_gclear:c { g_@@_color _ ##1 _tl }
5443             \pgfusepath { fill }
5444         \end { pgfscope }
5445     }
5446 }
5447 \endpgfpicture
5448 }

```

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

```

5449 \cs_new_protected:Npn \@@_color_opacity
5450 {
5451     \peek_meaning:NTF [
5452         { \@@_color_opacity:w }
5453         { \@@_color_opacity:w [ ] }
5454     }

```

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 currying.

```

5455 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5456 {
5457     \tl_clear:N \l_tmpa_tl
5458     \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.

```

5459     \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5460     \tl_if_empty:NTF \l_tmpb_tl
5461         { \@declaredcolor }
5462         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5463 }

```

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

```

5464 \keys_define:nn { nicematrix / color-opacity }
5465 {
5466     opacity .tl_set:N          = \l_tmpa_tl ,
5467     opacity .value_required:n = true
5468 }

5469 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5470 {
5471     \tl_set:Nn \l_@@_rows_tl { #1 }
5472     \tl_set:Nn \l_@@_cols_tl { #2 }
5473     \@@_cartesian_path:
5474 }

```

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

```

5475 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5476 {
5477     \tl_if_blank:nF { #2 }
5478     {

```

```

5479 \@@_add_to_colors_seq:en
5480 { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5481 { \@@_cartesian_color:nn { #3 } { - } }
5482 }
5483 }

```

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

```

5484 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5485 {
5486   \tl_if_blank:nF { #2 }
5487   {
5488     \@@_add_to_colors_seq:en
5489     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5490     { \@@_cartesian_color:nn { - } { #3 } }
5491   }
5492 }

```

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

```

5493 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5494 {
5495   \tl_if_blank:nF { #2 }
5496   {
5497     \@@_add_to_colors_seq:en
5498     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5499     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5500   }
5501 }

```

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

```

5502 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5503 {
5504   \tl_if_blank:nF { #2 }
5505   {
5506     \@@_add_to_colors_seq:en
5507     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5508     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5509   }
5510 }

```

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

```

5511 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5512 {
5513   \@@_cut_on_hyphen:w #1 \q_stop
5514   \tl_clear_new:N \l_@@_tmpc_tl
5515   \tl_clear_new:N \l_@@_tmpd_tl
5516   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5517   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5518   \@@_cut_on_hyphen:w #2 \q_stop
5519   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5520   \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`.

```

5521 \@@_cartesian_path:n { #3 }
5522 }

```

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

```

5523 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5524 {
5525   \clist_map_inline:nn { #3 }
5526   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5527 }

```

```

5528 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5529 {
5530   \int_step_inline:nn { \int_use:N \c@iRow }
5531   {
5532     \int_step_inline:nn { \int_use:N \c@jCol }
5533     {
5534       \int_if_even:nTF { ####1 + ##1 }
5535       { \@@_cellcolor [ #1 ] { #2 } }
5536       { \@@_cellcolor [ #1 ] { #3 } }
5537       { ##1 - ####1 }
5538     }
5539   }
5540 }

```

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”.

```

5541 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5542 {
5543   \@@_rectanglecolor [ #1 ] { #2 }
5544   { 1 - 1 }
5545   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5546 }

```

```

5547 \keys_define:nn { NiceMatrix / rowcolors }
5548 {
5549   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5550   respect-blocks .default:n = true ,
5551   cols .tl_set:N = \l_@@_cols_tl ,
5552   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5553   restart .default:n = true ,
5554   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5555 }

```

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*.

```

5556 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5557 {

```

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

```

5558   \group_begin:
5559   \seq_clear_new:N \l_@@_colors_seq
5560   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5561   \tl_clear_new:N \l_@@_cols_tl
5562   \tl_set:Nn \l_@@_cols_tl { - }
5563   \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).

```

5564   \int_zero_new:N \l_@@_color_int
5565   \int_set:Nn \l_@@_color_int 1
5566   \bool_if:NT \l_@@_respect_blocks_bool
5567   {

```

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`).

```

5568     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5569     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5570     { \@@_not_in_exterior_p:nnnnn ##1 }
5571   }
5572   \pgfpicture
5573   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5574   \clist_map_inline:nn { #2 }
5575   {
5576     \tl_set:Nn \l_tmpa_tl { ##1 }
5577     \tl_if_in:NnTF \l_tmpa_tl { - }
5578     { \@@_cut_on_hyphen:w ##1 \q_stop }
5579     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, `\l_tmpa_tl` and `\l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

5580     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5581     \int_set:Nn \l_@@_color_int
5582     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5583     \int_zero_new:N \l_@@_tmpc_int
5584     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5585     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5586     {

```

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

```

5587       \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

5588       \bool_if:NT \l_@@_respect_blocks_bool
5589       {
5590         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5591         { \@@_intersect_our_row_p:nnnnn #####1 }
5592         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5593       }
5594       \tl_set:Nx \l_@@_rows_tl
5595       { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5596       \tl_clear_new:N \l_@@_color_tl
5597       \tl_set:Nx \l_@@_color_tl
5598       {
5599         \@@_color_index:n
5600         {
5601           \int_mod:nn
5602           { \l_@@_color_int - 1 }
5603           { \seq_count:N \l_@@_colors_seq }
5604           + 1
5605         }
5606       }
5607       \tl_if_empty:NF \l_@@_color_tl
5608       {
5609         \@@_add_to_colors_seq:ee
5610         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5611         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5612       }
5613       \int_incr:N \l_@@_color_int
5614       \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5615     }
5616   }
5617   \endpgfpicture

```

```

5618 \group_end:
5619 }

```

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.

```

5620 \cs_new:Npn \@@_color_index:n #1
5621 {
5622   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5623   { \@@_color_index:n { #1 - 1 } }
5624   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5625 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5626 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5627 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5628 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5629 {
5630   \int_compare:nNnT { #3 } > \l_tmpb_int
5631   { \int_set:Nn \l_tmpb_int { #3 } }
5632 }

```

```

5633 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5634 {
5635   \bool_lazy_or:nnTF
5636   { \int_if_zero_p:n { #4 } }
5637   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5638   \prg_return_false:
5639   \prg_return_true:
5640 }

```

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

```

5641 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5642 {
5643   \bool_if:nTF
5644   {
5645     \int_compare_p:n { #1 <= \l_tmpa_int }
5646     &&
5647     \int_compare_p:n { \l_tmpa_int <= #3 }
5648   }
5649   \prg_return_true:
5650   \prg_return_false:
5651 }

```

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`).

```

5652 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5653 {
5654   \bool_lazy_any:nT
5655   {
5656     { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5657     { \dim_compare_p:nNn { #1 } = \c_zero_dim }

```

```

5658     { ! \@@_nocolor_used_bool }
5659   }
5660   {
5661     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5662     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5663   }

```

We begin the loop over the columns.

```

5664   \clist_map_inline:Nn \l_@@_cols_tl
5665   {
5666     \tl_set:Nn \l_tmpa_tl { ##1 }
5667     \tl_if_in:NnTF \l_tmpa_tl { - }
5668     { \@@_cut_on_hyphen:w ##1 \q_stop }
5669     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5670     \bool_lazy_or:nnT
5671     { \tl_if_blank_p:V \l_tmpa_tl }
5672     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5673     { \tl_set:Nn \l_tmpa_tl { 1 } }
5674     \bool_lazy_or:nnT
5675     { \tl_if_blank_p:V \l_tmpb_tl }
5676     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5677     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5678     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5679     { \tl_set:Nx \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5680     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl

```

If we decide to provide the commands `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors` in the code-before of a `\SubMatrix`, we will have to modify the following line, by adding a kind of offset. We will have also some other lines to modify.

```

5681     \@@_qpoint:n { col - \l_tmpa_tl }
5682     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5683     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5684     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5685     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5686     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5687   \clist_map_inline:Nn \l_@@_rows_tl
5688   {
5689     \tl_set:Nn \l_tmpa_tl { #####1 }
5690     \tl_if_in:NnTF \l_tmpa_tl { - }
5691     { \@@_cut_on_hyphen:w #####1 \q_stop }
5692     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5693     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5694     \tl_if_empty:NT \l_tmpb_tl
5695     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5696     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5697     { \tl_set:Nx \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`.

```

5698     \seq_if_in:NxF \l_@@_corners_cells_seq
5699     { \l_tmpa_tl - \l_@@_tmpc_tl }
5700     {
5701       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5702       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5703       \@@_qpoint:n { row - \l_tmpa_tl }
5704       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5705       \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5706       \cs_if_exist:cF
5707       { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5708       {
5709         \pgfpathrectanglecorners
5710         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }

```

```

5711             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5712         }
5713     }
5714 }
5715 }
5716 }

```

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`).

```

5717 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

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

```

5718 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5719 {
5720     \bool_set_true:N \@@_nocolor_used_bool
5721     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5722     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5723     \clist_map_inline:Nn \l_@@_rows_tl
5724     {
5725         \clist_map_inline:Nn \l_@@_cols_tl
5726         { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5727     }
5728 }

```

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`.

```

5729 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5730 {
5731     \clist_set_eq:NN \l_tmpa_clist #1
5732     \clist_clear:N #1
5733     \clist_map_inline:Nn \l_tmpa_clist
5734     {
5735         \tl_set:Nn \l_tmpa_tl { ##1 }
5736         \tl_if_in:NnTF \l_tmpa_tl { - }
5737         { \@@_cut_on_hyphen:w ##1 \q_stop }
5738         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5739     }
5740     \bool_lazy_or:nnT
5741     { \tl_if_blank_p:V \l_tmpa_tl }
5742     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5743     { \tl_set:Nn \l_tmpa_tl { 1 } }
5744     \bool_lazy_or:nnT
5745     { \tl_if_blank_p:V \l_tmpb_tl }
5746     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5747     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5748     \int_compare:nNnT \l_tmpb_tl > #2
5749     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5750     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5751     { \clist_put_right:Nn #1 { #####1 } }
5752 }

```

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

```

5753 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5754 {
5755     \@@_test_color_inside:

```



```

5756 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5757 {

```

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).

```

5758 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5759 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5760 }
5761 \ignorespaces
5762 }

```

When the user uses the key color-inside, the following command will be linked to \rowcolor in the tabular.

```

5763 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5764 {
5765 \@@_test_color_inside:
5766 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5767 {
5768 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5769 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5770 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5771 }
5772 \ignorespaces
5773 }

```

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).

```

5774 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5775 { \@@_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.

```

5776 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5777 {
5778 \@@_test_color_inside:
5779 \peek_remove_spaces:n
5780 { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5781 }

5782 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5783 {

```

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.

```

5784 \seq_gclear:N \g_tmpa_seq
5785 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5786 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5787 \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.

```

5788 \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5789 {
5790 { \int_use:N \c@iRow }
5791 { \exp_not:n { #1 } }

```

```

5792     { \exp_not:n { #2 } }
5793     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5794   }
5795 }

```

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`).

```

5796 \cs_new_protected:Npn \g_@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5797 {
5798   \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).

```

5799   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5800   {
5801     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5802     {
5803       \g_@@_rowlistcolors
5804       [ \exp_not:n { #2 } ]
5805       { #1 - \int_eval:n { \c@iRow - 1 } }
5806       { \exp_not:n { #3 } }
5807       [ \exp_not:n { #4 } ]
5808     }
5809   }
5810 }

```

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.

```

5811 \cs_new_protected:Npn \g_@@_clear_rowlistcolors_seq:
5812 {
5813   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5814   { \g_@@_rowlistcolors_tabular_ii:nnnn ##1 }
5815   \seq_gclear:N \g_@@_rowlistcolors_seq
5816 }

5817 \cs_new_protected:Npn \g_@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5818 {
5819   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5820   { \g_@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5821 }

```

The first mandatory argument of the command `\g_@@_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.

```

5822 \NewDocumentCommand \g_@@_columncolor_preamble { 0 { } m }
5823 {

```

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).

```

5824   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5825   {

```

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).

```

5826      \tl_gput_left:Nx \g_@@_pre_code_before_tl
5827      {
5828          \exp_not:N \columncolor [ #1 ]
5829          { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5830      }
5831  }
5832  }

5833 \hook_gput_code:nnn { begindocument } { . }
5834 {
5835     \IfPackageLoadedTF { colortbl }
5836     {
5837         \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5838         \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5839         \cs_new_protected:Npn \@@_revert_colortbl:
5840         {
5841             \hook_gput_code:nnn { env / tabular / begin } { . }
5842             {
5843                 \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5844                 \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5845             }
5846         }
5847     }
5848     { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5849 }

```

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`.

```

5850 \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`.

```

5851 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5852 {
5853     \int_if_zero:nTF \l_@@_first_col_int
5854     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5855     {
5856         \int_if_zero:nTF \c@jCol
5857         {
5858             \int_compare:nNf \c@iRow = { -1 }
5859             { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5860         }
5861         { \@@_OnlyMainNiceMatrix_i:n { #1 } }

```

```

5862     }
5863 }

```

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.

```

5864 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5865 {
5866   \int_if_zero:nF \c@iRow
5867   {
5868     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5869     {
5870       \int_compare:nNnT \c@jCol > \c_zero_int
5871       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5872     }
5873   }
5874 }

```

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.

```

5875 \keys_define:nn { NiceMatrix / Rules }
5876 {
5877   position .int_set:N = \l_@@_position_int ,
5878   position .value_required:n = true ,
5879   start .int_set:N = \l_@@_start_int ,
5880   start .initial:n = 1 ,
5881   end .code:n =
5882     \bool_lazy_or:nnTF
5883     { \tl_if_empty_p:n { #1 } }
5884     { \str_if_eq_p:nn { #1 } { last } }
5885     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5886     { \int_set:Nn \l_@@_end_int { #1 } }
5887 }

```

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.

```

5888 \keys_define:nn { NiceMatrix / RulesBis }
5889 {
5890   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5891   multiplicity .initial:n = 1 ,
5892   dotted .bool_set:N = \l_@@_dotted_bool ,
5893   dotted .initial:n = false ,
5894   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]`.

```

5895   color .code:n =

```

```

5896 \@@_set_CT@arc@:n { #1 }
5897 \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5898 color .value_required:n = true ,
5899 sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5900 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.

```

5901 tikz .code:n =
5902 \IfPackageLoadedTF { tikz }
5903 { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5904 { \@@_error:n { tikz~without~tikz } } ,
5905 tikz .value_required:n = true ,
5906 total-width .dim_set:N = \l_@@_rule_width_dim ,
5907 total-width .value_required:n = true ,
5908 width .meta:n = { total-width = #1 } ,
5909 unknown .code:n = \@@_error:n { Unknown~key~for~RulesBis }
5910 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5911 \cs_new_protected:Npn \@@_vline:n #1
5912 {

```

The group is for the options.

```

5913 \group_begin:
5914 \int_zero_new:N \l_@@_end_int
5915 \int_set_eq:NN \l_@@_end_int \c@iRow
5916 \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).

```

5917 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5918 \@@_vline_i:
5919 \group_end:
5920 }

```

```

5921 \cs_new_protected:Npn \@@_vline_i:
5922 {
5923 \int_zero_new:N \l_@@_local_start_int
5924 \int_zero_new:N \l_@@_local_end_int

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

5925 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5926 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5927 \l_tmpa_tl
5928 {

```

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.

```

5929 \bool_gset_true:N \g_tmpa_bool
5930 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5931 { \@@_test_vline_in_block:nnnnn ##1 }
5932 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5933 { \@@_test_vline_in_block:nnnnn ##1 }
5934 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5935 { \@@_test_vline_in_stroken_block:nnnn ##1 }
5936 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5937 \bool_if:NTF \g_tmpa_bool
5938 {
5939 \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.

```

5940         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5941     }
5942     {
5943         \int_compare:nNnT \l_@@_local_start_int > 0
5944         {
5945             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5946             \@@_vline_ii:
5947             \int_zero:N \l_@@_local_start_int
5948         }
5949     }
5950 }
5951 \int_compare:nNnT \l_@@_local_start_int > 0
5952 {
5953     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5954     \@@_vline_ii:
5955 }
5956 }

5957 \cs_new_protected:Npn \@@_test_in_corner_v:
5958 {
5959     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5960     {
5961         \seq_if_in:NxT
5962         \l_@@_corners_cells_seq
5963         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5964         { \bool_set_false:N \g_tmpa_bool }
5965     }
5966     {
5967         \seq_if_in:NxT
5968         \l_@@_corners_cells_seq
5969         { \l_tmpa_tl - \l_tmpb_tl }
5970         {
5971             \int_compare:nNnTF \l_tmpb_tl = 1
5972             { \bool_set_false:N \g_tmpa_bool }
5973             {
5974                 \seq_if_in:NxT
5975                 \l_@@_corners_cells_seq
5976                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5977                 { \bool_set_false:N \g_tmpa_bool }
5978             }
5979         }
5980     }
5981 }

5982 \cs_new_protected:Npn \@@_vline_ii:
5983 {
5984     \tl_clear:N \l_@@_tikz_rule_tl
5985     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5986     \bool_if:NTF \l_@@_dotted_bool
5987     \@@_vline_iv:
5988     {
5989         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5990         \@@_vline_iii:
5991         \@@_vline_v:
5992     }
5993 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

5994 \cs_new_protected:Npn \@@_vline_iii:

```

```

5995 {
5996   \pgfpicture
5997   \pgfrememberpicturepositiononpagetrue
5998   \pgf@relevantforpicturesizefalse
5999   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6000   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6001   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6002   \dim_set:Nn \l_tmpb_dim
6003     {
6004       \pgf@x
6005       - 0.5 \l_@@_rule_width_dim
6006       +
6007       ( \arrayrulewidth * \l_@@_multiplicity_int
6008         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6009     }
6010   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6011   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6012   \bool_lazy_all:nT
6013     {
6014       { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6015       { \cs_if_exist_p:N \CT@drsc@ }
6016       { ! \tl_if_blank_p:V \CT@drsc@ }
6017     }
6018     {
6019       \group_begin:
6020       \CT@drsc@
6021       \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6022       \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6023       \dim_set:Nn \l_@@_tmpd_dim
6024         {
6025           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6026           * ( \l_@@_multiplicity_int - 1 )
6027         }
6028       \pgfpathrectanglecorners
6029         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6030         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6031       \pgfusepath { fill }
6032       \group_end:
6033     }
6034   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6035   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6036   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6037     {
6038       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6039       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6040       \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6041       \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6042     }
6043   \CT@arc@
6044   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6045   \pgfsetrectcap
6046   \pgfusepathqstroke
6047   \endpgfpicture
6048 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6049 \cs_new_protected:Npn \@@_vline_iv:
6050 {
6051   \pgfpicture
6052   \pgfrememberpicturepositiononpagetrue
6053   \pgf@relevantforpicturesizefalse
6054   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6055   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }

```

```

6056 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6057 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6058 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6059 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6060 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6061 \CT@arc@
6062 \@@_draw_line:
6063 \endpgfpicture
6064 }

```

The following code is for the case when the user uses the key `tikz`.

```

6065 \cs_new_protected:Npn \@@_vline_v:
6066 {
6067   \begin {tikzpicture }
6068   % 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).

```

6069   \CT@arc@
6070   \tl_if_empty:NF \l_@@_rule_color_tl
6071   { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6072   \pgfrememberpicturepositiononpagetrue
6073   \pgf@relevantforpicturesizefalse
6074   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6075   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6076   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6077   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6078   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6079   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6080   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6081   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6082   ( \l_tmpb_dim , \l_tmpa_dim ) --
6083   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6084   \end {tikzpicture }
6085 }

```

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).

```

6086 \cs_new_protected:Npn \@@_draw_vlines:
6087 {
6088   \int_step_inline:nnn
6089   {
6090     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6091     1 2
6092   }
6093   {
6094     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6095     { \int_eval:n { \c@jCol + 1 } }
6096     \c@jCol
6097   }
6098   {
6099     \tl_if_eq:NnF \l_@@_vlines_clist { all }
6100     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6101     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6102   }
6103 }

```

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}`.


```

6104 \cs_new_protected:Npn \@@_hline:n #1
6105 {

```

The group is for the options.

```

6106   \group_begin:
6107   \int_zero_new:N \l_@@_end_int
6108   \int_set_eq:NN \l_@@_end_int \c@jCol
6109   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6110   \@@_hline_i:
6111   \group_end:
6112 }

6113 \cs_new_protected:Npn \@@_hline_i:
6114 {
6115   \int_zero_new:N \l_@@_local_start_int
6116   \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 .

```

6117   \tl_set:Nx \l\_tmpa\_tl { \int_use:N \l_@@_position_int }
6118   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6119     \l\_tmpb\_tl
6120   {

```

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.

```

6121     \bool_gset_true:N \g\_tmpa\_bool
6122     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6123       { \@@_test_hline_in_block:nnnnn ##1 }
6124     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6125       { \@@_test_hline_in_block:nnnnn ##1 }
6126     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6127       { \@@_test_hline_in_stroken_block:nnnn ##1 }
6128     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6129     \bool_if:NTF \g\_tmpa\_bool
6130     {
6131       \int_if_zero:nT \l_@@_local_start_int

```

We keep in memory that we have a rule to draw. \l_tmpc_tl will be the starting row of the rule that we will have to draw.

```

6132       { \int_set:Nn \l_@@_local_start_int \l\_tmpc\_tl }
6133     }
6134     {
6135       \int_compare:nNnT \l_@@_local_start_int > 0
6136       {
6137         \int_set:Nn \l_@@_local_end_int { \l\_tmpb\_tl - 1 }
6138         \@@_hline_ii:
6139         \int_zero:N \l_@@_local_start_int
6140       }
6141     }
6142   }
6143   \int_compare:nNnT \l_@@_local_start_int > 0
6144   {
6145     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6146     \@@_hline_ii:
6147   }
6148 }

```

```

6149 \cs_new_protected:Npn \@@_test_in_corner_h:
6150 {
6151   \int_compare:nNnTF \l\_tmpa\_tl = { \int_eval:n { \c@iRow + 1 } }
6152   {

```

```

6153     \seq_if_in:NxT
6154     \l_@@_corners_cells_seq
6155     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6156     { \bool_set_false:N \g_tmpa_bool }
6157   }
6158   {
6159     \seq_if_in:NxT
6160     \l_@@_corners_cells_seq
6161     { \l_tmpa_tl - \l_tmpb_tl }
6162     {
6163       \int_compare:nNnTF \l_tmpa_tl = 1
6164       { \bool_set_false:N \g_tmpa_bool }
6165       {
6166         \seq_if_in:NxT
6167         \l_@@_corners_cells_seq
6168         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6169         { \bool_set_false:N \g_tmpa_bool }
6170       }
6171     }
6172   }
6173 }

6174 \cs_new_protected:Npn \@@_hline_ii:
6175 {
6176   \tl_clear:N \l_@@_tikz_rule_tl
6177   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6178   \bool_if:NTF \l_@@_dotted_bool
6179   \@@_hline_iv:
6180   {
6181     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6182     \@@_hline_iii:
6183     \@@_hline_v:
6184   }
6185 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6186 \cs_new_protected:Npn \@@_hline_iii:
6187 {
6188   \pgfpicture
6189   \pgfrememberpicturepositiononpagetrue
6190   \pgf@relevantforpicturesizefalse
6191   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6192   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6193   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6194   \dim_set:Nn \l_tmpb_dim
6195   {
6196     \pgf@y
6197     - 0.5 \l_@@_rule_width_dim
6198     +
6199     ( \arrayrulewidth * \l_@@_multiplicity_int
6200       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6201   }
6202   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6203   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6204   \bool_lazy_all:nT
6205   {
6206     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6207     { \cs_if_exist_p:N \CT@drsc@ }
6208     { ! \tl_if_blank_p:V \CT@drsc@ }
6209   }
6210   {
6211     \group_begin:

```

```

6212 \CT@drsc@
6213 \dim_set:Nn \l_@@_tmpd_dim
6214 {
6215     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6216     * ( \l_@@_multiplicity_int - 1 )
6217 }
6218 \pgfpathrectanglecorners
6219 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6220 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6221 \pgfusepathqfill
6222 \group_end:
6223 }
6224 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6225 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6226 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6227 {
6228     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6229     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6230     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6231     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6232 }
6233 \CT@arc@
6234 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6235 \pgfsetrectcap
6236 \pgfusepathqstroke
6237 \endpgfpicture
6238 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6239 \cs_new_protected:Npn \@@_hline_iv:
6240 {
6241     \pgfpicture
6242     \pgfrememberpicturepositiononpagetrue
6243     \pgf@relevantforpicturesizefalse
6244     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6245     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6246     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6247     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6248     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6249     \int_compare:nNnT \l_@@_local_start_int = 1
6250     {
6251         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6252         \bool_if:NF \g_@@_delims_bool
6253         { \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.

```

6254     \tl_if_eq:NnF \g_@@_left_delim_tl (
6255       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6256     )
6257     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6258     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6259     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6260     {
6261       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6262       \bool_if:NF \g_@@_delims_bool
6263       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6264       \tl_if_eq:NnF \g_@@_right_delim_tl )
6265       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6266     }
6267     \CT@arc@
6268     \@@_draw_line:
6269     \endpgfpicture
6270   }

```

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`).

```

6271 \cs_new_protected:Npn \@@_hline_v:
6272 {
6273   \begin { tikzpicture }
6274   % 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).

```

6275     \CT@arc@
6276     \tl_if_empty:NF \l_@@_rule_color_tl
6277     { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6278     \pgfrememberpicturepositiononpagetrue
6279     \pgf@relevantforpicturesizefalse
6280     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6281     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6282     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6283     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6284     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6285     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6286     \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6287     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6288     ( \l_tmpa_dim , \l_tmpb_dim ) --
6289     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6290     \end { tikzpicture }
6291   }

```

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).

```

6292 \cs_new_protected:Npn \@@_draw_hlines:
6293 {
6294   \int_step_inline:nnn
6295   {
6296     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6297     1 2
6298   }
6299   {
6300     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6301     { \int_eval:n { \c@iRow + 1 } }
6302     \c@iRow

```

```

6303     }
6304     {
6305         \tl_if_eq:NnF \l_@@_hlines_clist { all }
6306         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6307         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6308     }
6309 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6310 \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.

```

6311 \cs_set:Npn \@@_Hline_i:n #1
6312 {
6313     \peek_remove_spaces:n
6314     {
6315         \peek_meaning:NTF \Hline
6316         { \@@_Hline_ii:nn { #1 + 1 } }
6317         { \@@_Hline_iii:n { #1 } }
6318     }
6319 }
6320 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6321 \cs_set:Npn \@@_Hline_iii:n #1
6322 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6323 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6324 {
6325     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6326     \skip_vertical:n { \l_@@_rule_width_dim }
6327     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6328     {
6329         \@@_hline:n
6330         {
6331             multiplicity = #1 ,
6332             position = \int_eval:n { \c@iRow + 1 } ,
6333             total-width = \dim_use:N \l_@@_rule_width_dim ,
6334             #2
6335         }
6336     }
6337     \egroup
6338 }

```

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`).

```

6339 \cs_new_protected:Npn \@@_custom_line:n #1
6340 {
6341     \str_clear_new:N \l_@@_command_str
6342     \str_clear_new:N \l_@@_ccommand_str
6343     \str_clear_new:N \l_@@_letter_str
6344     \tl_clear_new:N \l_@@_other_keys_tl
6345     \keys_set_known:nnN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6346     \bool_lazy_all:nTF
6347     {

```

```

6348     { \str_if_empty_p:N \l_@@_letter_str }
6349     { \str_if_empty_p:N \l_@@_command_str }
6350     { \str_if_empty_p:N \l_@@_ccommand_str }
6351   }
6352   { \@@_error:n { No~letter-and-no~command } }
6353   { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
6354 }

6355 \keys_define:nn { NiceMatrix / custom-line }
6356 {
6357   letter .str_set:N = \l_@@_letter_str ,
6358   letter .value_required:n = true ,
6359   command .str_set:N = \l_@@_command_str ,
6360   command .value_required:n = true ,
6361   ccommand .str_set:N = \l_@@_ccommand_str ,
6362   ccommand .value_required:n = true ,
6363 }

6364 \cs_new_protected:Npn \@@_custom_line_i:n #1
6365 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6366   \bool_set_false:N \l_@@_tikz_rule_bool
6367   \bool_set_false:N \l_@@_dotted_rule_bool
6368   \bool_set_false:N \l_@@_color_bool

6369   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6370   \bool_if:NT \l_@@_tikz_rule_bool
6371   {
6372     \IfPackageLoadedTF { tikz }
6373     { }
6374     { \@@_error:n { tikz~in~custom-line~without~tikz } }
6375     \bool_if:NT \l_@@_color_bool
6376     { \@@_error:n { color~in~custom-line~with~tikz } }
6377   }
6378   \bool_if:nT
6379   {
6380     \int_compare_p:nNn \l_@@_multiplicity_int > 1
6381     && \l_@@_dotted_rule_bool
6382   }
6383   { \@@_error:n { key~multiplicity~with~dotted } }
6384   \str_if_empty:NF \l_@@_letter_str
6385   {
6386     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6387     { \@@_error:n { Several~letters } }
6388     {
6389       \exp_args:NnV \tl_if_in:NnTF
6390       \c_@@_forbidden_letters_str \l_@@_letter_str
6391       { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6392       {

```

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.

```

6393       \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6394       { \@@_v_custom_line:n { #1 } }
6395     }
6396   }
6397 }
6398 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6399 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6400 }

```

```

6401 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6402 \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.

```

6403 \keys_define:nn { NiceMatrix / custom-line-bis }
6404 {
6405   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6406   multiplicity .initial:n = 1 ,
6407   multiplicity .value_required:n = true ,
6408   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6409   color .value_required:n = true ,
6410   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6411   tikz .value_required:n = true ,
6412   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6413   dotted .value_forbidden:n = true ,
6414   total-width .code:n = { } ,
6415   total-width .value_required:n = true ,
6416   width .code:n = { } ,
6417   width .value_required:n = true ,
6418   sep-color .code:n = { } ,
6419   sep-color .value_required:n = true ,
6420   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6421 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6422 \bool_new:N \l_@@_dotted_rule_bool
6423 \bool_new:N \l_@@_tikz_rule_bool
6424 \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`.

```

6425 \keys_define:nn { NiceMatrix / custom-line-width }
6426 {
6427   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6428   multiplicity .initial:n = 1 ,
6429   multiplicity .value_required:n = true ,
6430   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6431   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6432   \bool_set_true:N \l_@@_total_width_bool ,
6433   total-width .value_required:n = true ,
6434   width .meta:n = { total-width = #1 } ,
6435   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6436 }

```

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`).

```

6437 \cs_new_protected:Npn \@@_h_custom_line:n #1
6438 {

```

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`).

```

6439   \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6440   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6441 }

```

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`).

```
6442 \cs_new_protected:Npn \@@_c_custom_line:n #1
6443 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6444 \exp_args:Nc \NewExpandableDocumentCommand
6445 { \nicematrix - \l_@@_ccommand_str }
6446 { 0 { } m }
6447 {
6448 \noalign
6449 {
6450 \@@_compute_rule_width:n { #1 , ##1 }
6451 \skip_vertical:n { \l_@@_rule_width_dim }
6452 \clist_map_inline:nn
6453 { ##2 }
6454 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6455 }
6456 }
6457 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6458 }
```

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*.

```
6459 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6460 {
6461 \str_if_in:nnTF { #2 } { - }
6462 { \@@_cut_on_hyphen:w #2 \q_stop }
6463 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6464 \tl_gput_right:Nx \g_@@_pre_code_after_tl
6465 {
6466 \@@_hline:n
6467 {
6468 #1 ,
6469 start = \l_tmpa_tl ,
6470 end = \l_tmpb_tl ,
6471 position = \int_eval:n { \c@iRow + 1 } ,
6472 total-width = \dim_use:N \l_@@_rule_width_dim
6473 }
6474 }
6475 }

6476 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6477 {
6478 \bool_set_false:N \l_@@_tikz_rule_bool
6479 \bool_set_false:N \l_@@_total_width_bool
6480 \bool_set_false:N \l_@@_dotted_rule_bool
6481 \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6482 \bool_if:NF \l_@@_total_width_bool
6483 {
6484 \bool_if:NTF \l_@@_dotted_rule_bool
6485 { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6486 {
6487 \bool_if:NF \l_@@_tikz_rule_bool
6488 {
6489 \dim_set:Nn \l_@@_rule_width_dim
6490 {
6491 \arrayrulewidth * \l_@@_multiplicity_int
6492 + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6493 }
6494 }
6495 }
6496 }
6497 }
```



```

6498 \cs_new_protected:Npn \@@_v_custom_line:n #1
6499 {
6500   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6501   \tl_gput_right:Nx \g_@@_array_preamble_tl
6502     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6503   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6504     {
6505     \@@_vline:n
6506     {
6507       #1 ,
6508       position = \int_eval:n { \c@jCol + 1 } ,
6509       total-width = \dim_use:N \l_@@_rule_width_dim
6510     }
6511   }
6512   \@@_rec_preamble:n
6513 }
6514 \@@_custom_line:n
6515 { 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.

```

6516 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6517 {
6518   \bool_lazy_all:nT
6519   {
6520     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6521     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6522     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6523     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6524   }
6525   { \bool_gset_false:N \g_tmpa_bool }
6526 }

```

The same for vertical rules.

```

6527 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6528 {
6529   \bool_lazy_all:nT
6530   {
6531     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6532     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6533     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6534     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6535   }
6536   { \bool_gset_false:N \g_tmpa_bool }
6537 }
6538 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6539 {
6540   \bool_lazy_all:nT
6541   {
6542     {
6543       ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6544       || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6545     }
6546     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6547     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6548   }
6549   { \bool_gset_false:N \g_tmpa_bool }

```

```

6550 }
6551 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6552 {
6553   \bool_lazy_all:nT
6554   {
6555     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6556     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6557     {
6558       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6559       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6560     }
6561   }
6562   { \bool_gset_false:N \g_tmpa_bool }
6563 }

```

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.

```

6564 \cs_new_protected:Npn \@@_compute_corners:
6565 {

```

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.

```

6566   \seq_clear_new:N \l_@@_corners_cells_seq
6567   \clist_map_inline:Nn \l_@@_corners_clist
6568   {
6569     \str_case:nnF { ##1 }
6570     {
6571       { NW }
6572       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6573       { NE }
6574       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6575       { SW }
6576       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6577       { SE }
6578       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6579     }
6580     { \@@_error:nn { bad~corner } { ##1 } }
6581   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6582   \seq_if_empty:NF \l_@@_corners_cells_seq
6583   {

```

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.

```

6584     \tl_build_gput_right:Nx \g_@@_aux_tl
6585     {
6586       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6587       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6588     }
6589   }
6590 }

```

“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.

```
6591 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6592 {
```

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.

```
6593   \bool_set_false:N \l_tmpa_bool
6594   \int_zero_new:N \l_@@_last_empty_row_int
6595   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6596   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6597   {
6598     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6599     \bool_lazy_or:nnTF
6600     {
6601       \cs_if_exist_p:c
6602       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6603     }
6604     \l_tmpb_bool
6605     { \bool_set_true:N \l_tmpa_bool }
6606     {
6607       \bool_if:NF \l_tmpa_bool
6608       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6609     }
6610   }
```

Now, you determine the last empty cell in the row of number 1.

```
6611   \bool_set_false:N \l_tmpa_bool
6612   \int_zero_new:N \l_@@_last_empty_column_int
6613   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6614   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6615   {
6616     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6617     \bool_lazy_or:nnTF
6618     \l_tmpb_bool
6619     {
6620       \cs_if_exist_p:c
6621       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6622     }
6623     { \bool_set_true:N \l_tmpa_bool }
6624     {
6625       \bool_if:NF \l_tmpa_bool
6626       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6627     }
6628   }
```

Now, we loop over the rows.

```
6629   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6630   {
```

We treat the row number ##1 with another loop.

```

6631 \bool_set_false:N \l_tmpa_bool
6632 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6633 {
6634   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6635   \bool_lazy_or:nnTF
6636     \l_tmpb_bool
6637     {
6638       \cs_if_exist_p:c
6639         { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6640     }
6641   { \bool_set_true:N \l_tmpa_bool }
6642   {
6643     \bool_if:NF \l_tmpa_bool
6644     {
6645       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6646       \seq_put_right:Nn
6647         \l_@@_corners_cells_seq
6648         { ##1 - #####1 }
6649     }
6650   }
6651 }
6652 }
6653 }

```

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`).

```

6654 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6655 {
6656   \int_set:Nn \l_tmpa_int { #1 }
6657   \int_set:Nn \l_tmpb_int { #2 }
6658   \bool_set_false:N \l_tmpb_bool
6659   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6660     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6661 }
6662 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6663 {
6664   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6665   {
6666     \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6667     {
6668       \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6669       {
6670         \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6671         { \bool_set_true:N \l_tmpb_bool }
6672       }
6673     }
6674   }
6675 }

```

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.

```

6676 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6677 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6678 {
6679   auto-columns-width .code:n =
6680   {
6681     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6682     \dim_gzero_new:N \g_@@_max_cell_width_dim
6683     \bool_set_true:N \l_@@_auto_columns_width_bool
6684   }
6685 }

6686 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6687 {
6688   \int_gincr:N \g_@@_NiceMatrixBlock_int
6689   \dim_zero:N \l_@@_columns_width_dim
6690   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6691   \bool_if:NT \l_@@_block_auto_columns_width_bool
6692   {
6693     \cs_if_exist:cT
6694     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6695     {
6696       % is \exp_args:NNe mandatory?
6697       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6698       {
6699         \use:c
6700         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6701       }
6702     }
6703   }
6704 }

```

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`).

```

6705 {
6706   \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.

```

6707   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6708   {
6709     \bool_if:NT \l_@@_block_auto_columns_width_bool
6710     {
6711       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6712       \iow_shipout:Nx \@mainaux
6713       {
6714         \cs_gset:cpn
6715         { @@ _ 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.

```

6716         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6717       }
6718       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6719     }
6720   }
6721   \ignorespacesafterend
6722 }

```

26 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```
6723 \cs_generate_variant:Nn \dim_min:nn { v n }
6724 \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).

```
6725 \cs_new_protected:Npn \@@_create_extra_nodes:
6726 {
6727   \bool_if:nTF \l_@@_medium_nodes_bool
6728   {
6729     \bool_if:NTF \l_@@_large_nodes_bool
6730     \@@_create_medium_and_large_nodes:
6731     \@@_create_medium_nodes:
6732   }
6733   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6734 }
```

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`.

```
6735 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6736 {
6737   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6738   {
6739     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6740     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6741     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6742     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6743   }
6744   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6745   {
6746     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6747     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6748     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6749     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6750   }
}
```

We begin the two nested loops over the rows and the columns of the array.

```
6751 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6752 {
6753   \int_step_variable:nnNn
6754   \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.

```

6755     {
6756         \cs_if_exist:cT
6757         { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor `southwest` of the (normal) node of the cell (i - j). They will be stored in `\pgf@x` and `\pgf@y`.

```

6758     {
6759         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
6760         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6761         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6762         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6763         {
6764             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6765             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6766         }

```

We retrieve the coordinates of the anchor `northeast` of the (normal) node of the cell (i - j). They will be stored in `\pgf@x` and `\pgf@y`.

```

6767         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
6768         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6769         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6770         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6771         {
6772             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6773             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6774         }
6775     }
6776 }
6777 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6778 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6779 {
6780     \dim_compare:nNnT
6781     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6782     {
6783         \@@_qpoint:n { row - \@@_i: - base }
6784         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6785         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6786     }
6787 }
6788 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6789 {
6790     \dim_compare:nNnT
6791     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6792     {
6793         \@@_qpoint:n { col - \@@_j: }
6794         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6795         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6796     }
6797 }
6798 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6799 \cs_new_protected:Npn \@@_create_medium_nodes:
6800 {
6801     \pgfpicture
6802     \pgfrememberpicturepositiononpagetrue
6803     \pgf@relevantforpicturesizefalse
6804     \@@_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”.

```

6805     \tl_set:Nn \l_@@_suffix_tl { -medium }
6806     \@@_create_nodes:
6807     \endpgfpicture
6808 }

```

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:`.

```

6809 \cs_new_protected:Npn \@@_create_large_nodes:
6810 {
6811   \pgfpicture
6812   \pgfrememberpicturepositiononpagetrue
6813   \pgf@relevantforpicturesizefalse
6814   \@@_computations_for_medium_nodes:
6815   \@@_computations_for_large_nodes:
6816   \tl_set:Nn \l_@@_suffix_tl { - large }
6817   \@@_create_nodes:
6818   \endpgfpicture
6819 }
6820 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6821 {
6822   \pgfpicture
6823   \pgfrememberpicturepositiononpagetrue
6824   \pgf@relevantforpicturesizefalse
6825   \@@_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”.

```

6826     \tl_set:Nn \l_@@_suffix_tl { - medium }
6827     \@@_create_nodes:
6828     \@@_computations_for_large_nodes:
6829     \tl_set:Nn \l_@@_suffix_tl { - large }
6830     \@@_create_nodes:
6831     \endpgfpicture
6832 }

```

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.

```

6833 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6834 {
6835   \int_set:Nn \l_@@_first_row_int 1
6836   \int_set:Nn \l_@@_first_col_int 1

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

6837   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6838   {
6839     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6840     {
6841       (
6842         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6843         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6844       )
6845       / 2
6846     }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`


```

6847     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6848     { l_@@_row_ \@@_i: _ min_dim }
6849   }
6850   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6851   {
6852     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6853     {
6854       (
6855         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6856         \dim_use:c
6857         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6858       )
6859       / 2
6860     }
6861     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6862     { l_@@_column _ \@@_j: _ max _ dim }
6863   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6864   \dim_sub:cn
6865   { l_@@_column _ 1 _ min _ dim }
6866   \l_@@_left_margin_dim
6867   \dim_add:cn
6868   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6869   \l_@@_right_margin_dim
6870 }

```

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).

```

6871 \cs_new_protected:Npn \@@_create_nodes:
6872 {
6873   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6874   {
6875     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6876     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6877     \@@_pgf_rect_node:nnnnn
6878     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6879     { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
6880     { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
6881     { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
6882     { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
6883     \str_if_empty:NF \l_@@_name_str
6884     {
6885       \pgfnodealias
6886       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6887       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6888     }
6889   }
6890 }

```

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 .

```

6891   \seq_map_pairwise_function:NNN
6892   \g_@@_multicolumn_cells_seq
6893   \g_@@_multicolumn_sizes_seq
6894   \@@_node_for_multicolumn:nn
6895 }

```

```

6896 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6897 {
6898   \cs_set_nopar:Npn \@@_i: { #1 }
6899   \cs_set_nopar:Npn \@@_j: { #2 }
6900 }

```

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”).

```

6901 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6902 {
6903   \@@_extract_coords_values: #1 \q_stop
6904   \@@_pgf_rect_node:nnnnn
6905   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6906   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6907   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6908   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6909   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6910   \str_if_empty:NF \l_@@_name_str
6911   {
6912     \pgfnodealias
6913     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6914     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6915   }
6916 }

```

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.

```

6917 \keys_define:nn { NiceMatrix / Block / FirstPass }
6918 {
6919   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6920   l .value_forbidden:n = true ,
6921   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6922   r .value_forbidden:n = true ,
6923   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6924   c .value_forbidden:n = true ,
6925   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6926   L .value_forbidden:n = true ,
6927   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6928   R .value_forbidden:n = true ,
6929   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6930   C .value_forbidden:n = true ,
6931   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6932   t .value_forbidden:n = true ,
6933   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6934   T .value_forbidden:n = true ,
6935   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6936   b .value_forbidden:n = true ,
6937   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6938   B .value_forbidden:n = true ,
6939   color .code:n =
6940     \@@_color:n { #1 }
6941     \tl_set_rescan:Nnn
6942     \l_@@_draw_tl

```

```

6943     { \char_set_catcode_other:N ! }
6944     { #1 } ,
6945     color .value_required:n = true ,
6946     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6947     respect-arraystretch .default:n = true
6948 }

```

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.

```

6949 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

6950 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6951 {

```

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).

```

6952   \peek_remove_spaces:n
6953   {
6954     \tl_if_blank:nTF { #2 }
6955     { \@@_Block_i 1-1 \q_stop }
6956     {
6957       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6958       \@@_Block_i_czech \@@_Block_i
6959       #2 \q_stop
6960     }
6961     { #1 } { #3 } { #4 }
6962   }
6963 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6964 \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`.

```

6965 {
6966   \char_set_catcode_active:N -
6967   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6968 }

```

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.

```

6969 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6970 {

```

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).

```

6971   \bool_lazy_or:nnTF
6972   { \tl_if_blank_p:n { #1 } }
6973   { \str_if_eq_p:nn { #1 } { * } }
6974   { \int_set:Nn \l_tmpa_int { 100 } }
6975   { \int_set:Nn \l_tmpa_int { #1 } }
6976   \bool_lazy_or:nnTF
6977   { \tl_if_blank_p:n { #2 } }
6978   { \str_if_eq_p:nn { #2 } { * } }
6979   { \int_set:Nn \l_tmpb_int { 100 } }
6980   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6981 \int_compare:nNnTF \l_tmpb_int = 1
6982 {
6983   \str_if_empty:NTF \l_@@_hpos_cell_str
6984   { \str_set:Nn \l_@@_hpos_block_str c }
6985   { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6986 }
6987 { \str_set:Nn \l_@@_hpos_block_str c }

```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```

6988 \keys_set_known:n { NiceMatrix / Block / FirstPass } { #3 }
6989 \tl_set:Nx \l_tmpa_tl
6990 {
6991   { \int_use:N \c@iRow }
6992   { \int_use:N \c@jCol }
6993   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6994   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6995 }

```

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).

```

6996 \bool_if:nTF
6997 {
6998   (
6999     \int_compare_p:nNn { \l_tmpa_int } = 1
7000     ||
7001     \int_compare_p:nNn { \l_tmpb_int } = 1
7002   )
7003   && ! \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.

```

7004   && ! \l_@@_X_bool
7005 }
7006 { \exp_args:Nee \@@_Block_iv:nnnnn }
7007 { \exp_args:Nee \@@_Block_v:nnnnn }
7008 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7009 }

```

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.

```

7010 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7011 {
7012   \int_gincr:N \g_@@_block_box_int
7013   \cs_set_protected_nopar:Npn \diagbox ##1 ##2

```

```

7014 {
7015   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7016   {
7017     \@@_actually_diagbox:nnnnnn
7018     { \int_use:N \c@iRow }
7019     { \int_use:N \c@jCol }
7020     { \int_eval:n { \c@iRow + #1 - 1 } }
7021     { \int_eval:n { \c@jCol + #2 - 1 } }
7022     { \g_@@_row_style_tl \exp_not:n { ##1 } }
7023     { \g_@@_row_style_tl \exp_not:n { ##2 } }
7024   }
7025 }
7026 \box_gclear_new:c
7027 { \g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7028 \hbox_gset:cn
7029 { \g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }
7030 {

```

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}`).

```

7031   \tl_if_empty:NTF \l_@@_color_tl
7032   { \int_compare:nNnT { #2 } = 1 \set@color }
7033   { \@@_color:V \l_@@_color_tl }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

7034   \int_compare:nNnT { #1 } = 1
7035   {
7036     \int_if_zero:nTF \c@iRow
7037     \l_@@_code_for_first_row_tl
7038     {
7039       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7040       \l_@@_code_for_last_row_tl
7041     }
7042     \g_@@_row_style_tl
7043   }
7044   \bool_if:NF \l_@@_respect_arraystretch_bool
7045   { \cs_set:Npn \arraystretch { 1 } }
7046   \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7047   #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.).

```

7048   \@@_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}`.

```

7049   \bool_if:NTF \l_@@_tabular_bool
7050   {
7051     \bool_lazy_all:nTF
7052     {
7053       { \int_compare_p:nNn { #2 } = 1 }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```

7054         { \dim_compare:n { \l_@@_col_width_dim >= \c_zero_dim } }
7055         { ! \g_@@_rotate_bool }
7056     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7057     {
7058         \use:e
7059         {
7060             \exp_not:N \begin { minipage }%
7061             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7062             { \l_@@_col_width_dim }
7063             \str_case:Vn \l_@@_hpos_block_str
7064             { c \centering r \raggedleft l \raggedright }
7065         }
7066         #5
7067         \end { minipage }
7068     }

```

In the other cases, we use a `{tabular}`.

```

7069     {
7070         \use:e
7071         {
7072             \exp_not:N \begin { tabular }%
7073             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7074             { @ { } \l_@@_hpos_block_str @ { } }
7075         }
7076         #5
7077         \end { tabular }
7078     }
7079 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7080     {
7081         \c_math_toggle_token
7082         \use:e
7083         {
7084             \exp_not:N \begin { array }%
7085             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7086             { @ { } \l_@@_hpos_block_str @ { } }
7087         }
7088         #5
7089         \end { array }
7090         \c_math_toggle_token
7091     }
7092 }

```

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).

```

7093     \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.

```

7094     \int_compare:nNnT { #2 } = 1
7095     {
7096         \dim_gset:Nn \g_@@_blocks_wd_dim
7097         {
7098             \dim_max:nn
7099             \g_@@_blocks_wd_dim
7100             {
7101                 \box_wd:c

```

```

7102         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7103     }
7104 }
7105 }

```

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.

```

7106 \str_if_eq:VnT \l_@@_vpos_of_block_str { c }
7107 {
7108     \int_compare:nNnT { #1 } = 1
7109     {
7110         \dim_gset:Nn \g_@@_blocks_ht_dim
7111         {
7112             \dim_max:nn
7113             \g_@@_blocks_ht_dim
7114             {
7115                 \box_ht:c
7116                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7117             }
7118         }
7119         \dim_gset:Nn \g_@@_blocks_dp_dim
7120         {
7121             \dim_max:nn
7122             \g_@@_blocks_dp_dim
7123             {
7124                 \box_dp:c
7125                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7126             }
7127         }
7128     }
7129 }
7130 \seq_gput_right:Nx \g_@@_blocks_seq
7131 {
7132     \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.

```

7133 {
7134     \exp_not:n { #3 } ,
7135     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7136     \bool_if:NT \g_@@_rotate_bool
7137     {
7138         \bool_if:NTF \g_@@_rotate_c_bool
7139         { v-center }
7140         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7141     }
7142 }
7143 {
7144     \box_use_drop:c
7145     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7146 }
7147 }
7148 \bool_set_false:N \g_@@_rotate_c_bool
7149 }
7150 }

```

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.

```

7151 \cs_new:Npn \@@_adjust_hpos_rotate:
7152 {
7153   \bool_if:NT \g_@@_rotate_bool
7154   {
7155     \str_set:Nx \l_@@_hpos_block_str
7156     {
7157       \bool_if:NTF \g_@@_rotate_c_bool
7158       { c }
7159       {
7160         \str_case:VnF \l_@@_vpos_of_block_str
7161         { b l B l t r T r }
7162         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7163       }
7164     }
7165   }
7166 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7167 \cs_new_protected:Npn \@@_rotate_box_of_block:
7168 {
7169   \box_grotate:cn
7170   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7171   { 90 }
7172   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7173   {
7174     \vbox_gset_top:cn
7175     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7176     {
7177       \skip_vertical:n { 0.8 ex }
7178       \box_use:c
7179       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7180     }
7181   }
7182   \bool_if:NT \g_@@_rotate_c_bool
7183   {
7184     \hbox_gset:cn
7185     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7186     {
7187       \c_math_toggle_token
7188       \vcenter
7189       {
7190         \box_use:c
7191         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7192       }
7193       \c_math_toggle_token
7194     }
7195   }
7196 }

```

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:nnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7197 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7198 {
7199   \seq_gput_right:Nx \g_@@_blocks_seq
7200   {
7201     \l_tmpa_tl

```



```

7202 { \exp_not:n { #3 } }
7203 {
7204   \bool_if:NTF \l_@@_tabular_bool
7205   {
7206     \group_begin:
7207     \bool_if:NF \l_@@_respect_arraystretch_bool
7208     { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7209     \exp_not:n
7210     {
7211       \dim_zero:N \extrarowheight
7212       #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).

```

7213       % \@@_adjust_hpos_rotate:
7214       \use:e
7215       {
7216         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
7217         { @ { } \l_@@_hpos_block_str @ { } }
7218       }
7219       #5
7220       \end { tabular }
7221     }
7222   \group_end:
7223 }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7224 {
7225   \group_begin:
7226   \bool_if:NF \l_@@_respect_arraystretch_bool
7227   { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7228   \exp_not:n
7229   {
7230     \dim_zero:N \extrarowheight
7231     #4
7232     % \@@_adjust_hpos_rotate:
7233     \c_math_toggle_token % :n c
7234     \use:e
7235     {
7236       \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
7237       { @ { } \l_@@_hpos_block_str @ { } }
7238     }
7239     #5
7240     \end { array }
7241     \c_math_toggle_token
7242   }
7243   \group_end:
7244 }
7245 }
7246 }
7247 }

```

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).

```

7248 \keys_define:nn { NiceMatrix / Block / SecondPass }
7249 {
7250   tikz .code:n =
7251     \IfPackageLoadedTF { tikz }
7252     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }

```

```

7253     { \@@_error:n { tikz~key~without~tikz } } ,
7254     tikz .value_required:n = true ,
7255     fill .code:n =
7256       \tl_set_rescan:Nnn
7257       \l_@@_fill_tl
7258       { \char_set_catcode_other:N ! }
7259       { #1 } ,
7260     fill .value_required:n = true ,
7261     opacity .tl_set:N = \l_@@_opacity_tl ,
7262     opacity .value_required:n = true ,
7263     draw .code:n =
7264       \tl_set_rescan:Nnn
7265       \l_@@_draw_tl
7266       { \char_set_catcode_other:N ! }
7267       { #1 } ,
7268     draw .default:n = default ,
7269     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7270     rounded-corners .default:n = 4 pt ,
7271     color .code:n =
7272       \@@_color:n { #1 }
7273       \tl_set_rescan:Nnn
7274       \l_@@_draw_tl
7275       { \char_set_catcode_other:N ! }
7276       { #1 } ,
7277     borders .clist_set:N = \l_@@_borders_clist ,
7278     borders .value_required:n = true ,
7279     hvlines .meta:n = { vlines , hlines } ,
7280     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7281     vlines .default:n = true ,
7282     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7283     hlines .default:n = true ,
7284     line-width .dim_set:N = \l_@@_line_width_dim ,
7285     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7286     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7287     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7288     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7289     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7290       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7291     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7292       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7293     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7294       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7295     t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
7296     T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
7297     b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
7298     B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
7299     v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
7300     v-center .value_forbidden:n = true ,
7301     name .tl_set:N = \l_@@_block_name_str ,
7302     name .value_required:n = true ,
7303     name .initial:n = ,
7304     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
7305     transparent .bool_set:N = \l_@@_transparent_bool ,
7306     transparent .default:n = true ,
7307     transparent .initial:n = false ,
7308     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7309   }

```

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.

```

7310 \cs_new_protected:Npn \@@_draw_blocks:
7311 {
7312   \cs_set_eq:NN \ialign \@@_old_ialign:
7313   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7314 }
7315 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7316 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7317   \int_zero_new:N \l_@@_last_row_int
7318   \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.

```

7319   \int_compare:nNnTF { #3 } > { 99 }
7320   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7321   { \int_set:Nn \l_@@_last_row_int { #3 } }
7322   \int_compare:nNnTF { #4 } > { 99 }
7323   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7324   { \int_set:Nn \l_@@_last_col_int { #4 } }
7325   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7326   {
7327     \bool_lazy_and:nnTF
7328       \l_@@_preamble_bool
7329       {
7330         \int_compare_p:n
7331           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7332       }
7333       {
7334         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7335         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7336         \@@_msg_redirect_name:nn { columns-not-used } { none }
7337       }
7338       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7339   }
7340   {
7341     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7342     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7343     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7344   }
7345 }

```

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

```

7346 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7347 {

```

The group is for the keys.

```

7348   \group_begin:
7349   \int_compare:nNnT { #1 } = { #3 }
7350   { \str_set:Nn \l_@@_vpos_of_block_str { t } }
7351   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7352   \bool_if:NT \l_@@_vlines_block_bool
7353   {
7354     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7355     {
7356       \@@_vlines_block:nnn

```

```

7357         { \exp_not:n { #5 } }
7358         { #1 - #2 }
7359         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7360     }
7361 }
7362 \bool_if:NT \l_@@_hlines_block_bool
7363 {
7364     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7365     {
7366         \@@_hlines_block:nnn
7367         { \exp_not:n { #5 } }
7368         { #1 - #2 }
7369         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7370     }
7371 }
7372 \bool_if:nF
7373 {
7374     \l_@@_transparent_bool
7375     || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7376 }
7377 {

```

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).

```

7378     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7379     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7380 }

7381 \bool_lazy_and:nnT
7382 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
7383 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7384 { \@@_error:n { hlines-with-color } }

7385 \tl_if_empty:NF \l_@@_draw_tl
7386 {
7387     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7388     {
7389         \@@_stroke_block:nnn
7390         { \exp_not:n { #5 } } % #5 are the options
7391         { #1 - #2 }
7392         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7393     }
7394     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7395     { { #1 } { #2 } { #3 } { #4 } }
7396 }

7397 \clist_if_empty:NF \l_@@_borders_clist
7398 {
7399     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7400     {
7401         \@@_stroke_borders_block:nnn
7402         { \exp_not:n { #5 } }
7403         { #1 - #2 }
7404         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7405     }
7406 }

7407 \tl_if_empty:NF \l_@@_fill_tl
7408 {
7409     \tl_if_empty:NF \l_@@_opacity_tl
7410     {
7411         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7412             {

```

```

7413         \tl_set:Nx \l_@@_fill_tl
7414         {
7415             [ opacity = \l_@@_opacity_tl ,
7416             \tl_tail:V \l_@@_fill_tl
7417         }
7418     }
7419     {
7420         \tl_set:Nx \l_@@_fill_tl
7421         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7422     }
7423 }
7424 \tl_gput_right:Nx \g_@@_pre_code_before_tl
7425 {
7426     \exp_not:N \roundedrectanglecolor
7427     \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7428         { \l_@@_fill_tl }
7429         { { \l_@@_fill_tl } }
7430         { #1 - #2 }
7431         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7432         { \dim_use:N \l_@@_rounded_corners_dim }
7433     }
7434 }
7435 \seq_if_empty:NF \l_@@_tikz_seq
7436 {
7437     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7438     {
7439         \@@_block_tikz:nnnnn
7440         { #1 }
7441         { #2 }
7442         { \int_use:N \l_@@_last_row_int }
7443         { \int_use:N \l_@@_last_col_int }
7444         { \seq_use:Nn \l_@@_tikz_seq { , } }
7445     }
7446 }
7447 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7448 {
7449     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7450     {
7451         \@@_actually_diagbox:nnnnn
7452         { #1 }
7453         { #2 }
7454         { \int_use:N \l_@@_last_row_int }
7455         { \int_use:N \l_@@_last_col_int }
7456         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7457     }
7458 }
7459 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7460 \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
three	four	two
six	seven	five
		eight

We highlight the node 1-1-block-short

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7461 \pgfpicture
7462 \pgfrememberpicturepositiononpagetrue
7463 \pgf@relevantforpicturesizefalse
7464 \@@_qpoint:n { row - #1 }
7465 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7466 \@@_qpoint:n { col - #2 }
7467 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7468 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7469 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7470 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7471 \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.

```

7472 \@@_pgf_rect_node:nnnnn
7473 { \@@_env: - #1 - #2 - block }
7474 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7475 \str_if_empty:NF \l_@@_block_name_str
7476 {
7477 \pgfnodealias
7478 { \@@_env: - \l_@@_block_name_str }
7479 { \@@_env: - #1 - #2 - block }
7480 \str_if_empty:NF \l_@@_name_str
7481 {
7482 \pgfnodealias
7483 { \l_@@_name_str - \l_@@_block_name_str }
7484 { \@@_env: - #1 - #2 - block }
7485 }
7486 }

```

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.

```

7487 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7488 {
7489 \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.

```

7490 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7491 {

```

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.

```

7492 \cs_if_exist:cT
7493 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7494 {
7495 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7496 {
7497 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7498 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7499 }
7500 }
7501 }

```

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.

```

7502     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7503     {
7504         \@@_qpoint:n { col - #2 }
7505         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7506     }
7507     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7508     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7509     {
7510         \cs_if_exist:cT
7511         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7512         {
7513             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7514             {
7515                 \pgfpointanchor
7516                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7517                 { east }
7518                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7519             }
7520         }
7521     }
7522     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7523     {
7524         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7525         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7526     }
7527     \@@_pgf_rect_node:nnnnn
7528     { \@@_env: - #1 - #2 - block - short }
7529     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7530 }

```

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.

```

7531     \bool_if:NT \l_@@_medium_nodes_bool
7532     {
7533         \@@_pgf_rect_node:nnn
7534         { \@@_env: - #1 - #2 - block - medium }
7535         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7536         {
7537             \pgfpointanchor
7538             { \@@_env:
7539               - \int_use:N \l_@@_last_row_int
7540               - \int_use:N \l_@@_last_col_int - medium
7541             }
7542             { south-east }
7543         }
7544     }

```

Now, we will put the label of the block.

```

7545     \bool_lazy_any:nTF
7546     {
7547         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7548         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7549         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7550     }
7551     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7552     \int_if_zero:nT { #2 } { \str_set:Nn \l_@@_hpos_block_str r }

```

If we are in the last column, we must put the block as if it was with the key 1.

```

7553     \bool_if:nT \g_@@_last_col_found_bool
7554     {
7555         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7556         { \str_set:Nn \l_@@_hpos_block_str 1 }
7557     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7558     \tl_set:Nx \l_tmpa_tl
7559     {
7560         \str_case:Vn \l_@@_vpos_of_block_str
7561         {
7562             c {
7563                 \str_case:Vn \l_@@_hpos_block_str
7564                 {
7565                     c { center }
7566                     l { west }
7567                     r { east }
7568                 }
7569             }
7570             T {
7571                 \str_case:Vn \l_@@_hpos_block_str
7572                 {
7573                     c { north }
7574                     l { north~west }
7575                     r { north~east }
7576                 }
7577             }
7578             B {
7579                 \str_case:Vn \l_@@_hpos_block_str
7580                 {
7581                     c { south }
7582                     l { south~west }
7583                     r { south~east }
7584                 }
7585             }
7586         }
7587     }
7588 }
7589 }
7590 }
7591 \pgftransformshift
7592 {
7593     \pgfpointanchor
7594     {
7595         \@@_env: - #1 - #2 - block
7596         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7597     }
7598     { \l_tmpa_tl }
7599 }
7600 \pgfset
7601 {
7602     inner~xsep = \c_zero_dim ,
7603     inner~ysep = \l_@@_block_ysep_dim
7604 }
7605 \pgfnode
7606 { rectangle }
7607 { \l_tmpa_tl }
7608 { \box_use_drop:N \l_@@_cell_box } { } { }
7609 }

```

End of the case when `\l_@@_vpos_of_block_str` is equal to c, T or B. Now, the other cases.

```

7610 {

```



```

7611 \pgfextracty \l_tmpa_dim
7612 {
7613   \@@_qpoint:n
7614   {
7615     row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7616     - base
7617   }
7618 }
7619 \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.

```

7620 \pgfpointanchor
7621 {
7622   \@@_env: - #1 - #2 - block
7623   \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7624 }
7625 {
7626   \str_case:Vn \l_@@_hpos_block_str
7627   {
7628     c { center }
7629     l { west }
7630     r { east }
7631   }
7632 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7633 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7634 \pgfset { inner~sep = \c_zero_dim }
7635 \pgfnode
7636 { rectangle }
7637 {
7638   \str_case:Vn \l_@@_hpos_block_str
7639   {
7640     c { base }
7641     l { base~west }
7642     r { base~east }
7643   }
7644 }
7645 { \box_use_drop:N \l_@@_cell_box } { } { }
7646 }
7647 \endpgfpicture
7648 \group_end:
7649 }

```

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).

```

7650 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7651 {
7652   \group_begin:
7653   \tl_clear:N \l_@@_draw_tl
7654   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7655   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7656   \pgfpicture
7657   \pgfrememberpicturepositiononpagetrue
7658   \pgf@relevantforpicturesizefalse
7659   \tl_if_empty:NF \l_@@_draw_tl
7660   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7661   \str_if_eq:VnTF \l_@@_draw_tl { default }
7662   { \CT@arc@ }

```

```

7663     { \l_@@_color:V \l_@@_draw_tl }
7664   }
7665   \pgfsetcornersarced
7666   {
7667     \pgfpoint
7668     { \l_@@_rounded_corners_dim }
7669     { \l_@@_rounded_corners_dim }
7670   }
7671   \l_@@_cut_on_hyphen:w #2 \q_stop
7672   \bool_lazy_and:nnT
7673   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7674   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7675   {
7676     \l_@@_qpoint:n { row - \l_tmpa_tl }
7677     \dim_set_eq:NN \l_tmpb_dim \pgf@y
7678     \l_@@_qpoint:n { col - \l_tmpb_tl }
7679     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7680     \l_@@_cut_on_hyphen:w #3 \q_stop
7681     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7682     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7683     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7684     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7685     \l_@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7686     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7687     \l_@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7688     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7689     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7690     \pgfpathrectanglecorners
7691     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7692     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7693     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7694     { \pgfusepathqstroke }
7695     { \pgfusepath { stroke } }
7696   }
7697   \endpgfpicture
7698   \group_end:
7699 }

```

Here is the set of keys for the command `\l_@@_stroke_block:nnn`.

```

7700 \keys_define:nn { NiceMatrix / BlockStroke }
7701 {
7702   color .tl_set:N = \l_@@_draw_tl ,
7703   draw .code:n =
7704     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7705   draw .default:n = default ,
7706   line-width .dim_set:N = \l_@@_line_width_dim ,
7707   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7708   rounded-corners .default:n = 4 pt
7709 }

```

The first argument of `\l_@@_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).

```

7710 \cs_new_protected:Npn \l_@@_vlines_block:nnn #1 #2 #3
7711 {
7712   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7713   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7714   \l_@@_cut_on_hyphen:w #2 \q_stop
7715   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7716   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7717   \l_@@_cut_on_hyphen:w #3 \q_stop
7718   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7719   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }

```

```

7720 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7721 {
7722   \use:e
7723   {
7724     \@@_vline:n
7725     {
7726       position = ##1 ,
7727       start = \l_@@_tmpc_tl ,
7728       end = \int_eval:n { \l_tmpa_tl - 1 } ,
7729       total-width = \dim_use:N \l_@@_line_width_dim
7730     }
7731   }
7732 }
7733 }
7734 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7735 {
7736   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7737   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7738   \@@_cut_on_hyphen:w #2 \q_stop
7739   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7740   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7741   \@@_cut_on_hyphen:w #3 \q_stop
7742   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7743   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7744   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7745   {
7746     \use:e
7747     {
7748       \@@_hline:n
7749       {
7750         position = ##1 ,
7751         start = \l_@@_tmpd_tl ,
7752         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7753         total-width = \dim_use:N \l_@@_line_width_dim
7754       }
7755     }
7756   }
7757 }

```

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).

```

7758 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7759 {
7760   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7761   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7762   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7763   { \@@_error:n { borders~forbidden } }
7764   {
7765     \tl_clear_new:N \l_@@_borders_tikz_tl
7766     \keys_set:nV
7767       { NiceMatrix / OnlyForTikzInBorders }
7768     \l_@@_borders_clist
7769     \@@_cut_on_hyphen:w #2 \q_stop
7770     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7771     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7772     \@@_cut_on_hyphen:w #3 \q_stop
7773     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7774     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7775     \@@_stroke_borders_block_i:
7776   }
7777 }

```

```

7778 \hook_gput_code:nnn { begindocument } { . }
7779 {
7780   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7781   {
7782     \c_@@_pgfortikzpicture_tl
7783     \@@_stroke_borders_block_ii:
7784     \c_@@_endpgfortikzpicture_tl
7785   }
7786 }
7787 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7788 {
7789   \pgfrememberpicturepositiononpagetrue
7790   \pgf@relevantforpicturesizefalse
7791   \CT@arc@
7792   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7793   \clist_if_in:NnT \l_@@_borders_clist { right }
7794   { \@@_stroke_vertical:n \l_tmpb_tl }
7795   \clist_if_in:NnT \l_@@_borders_clist { left }
7796   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7797   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7798   { \@@_stroke_horizontal:n \l_tmpa_tl }
7799   \clist_if_in:NnT \l_@@_borders_clist { top }
7800   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7801 }
7802 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7803 {
7804   tikz .code:n =
7805     \cs_if_exist:NTF \tikzpicture
7806     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7807     { \@@_error:n { tikz~in~borders~without~tikz } } ,
7808   tikz .value_required:n = true ,
7809   top .code:n = ,
7810   bottom .code:n = ,
7811   left .code:n = ,
7812   right .code:n = ,
7813   unknown .code:n = \@@_error:n { bad~border }
7814 }

```

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).

```

7815 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7816 {
7817   \@@_qpoint:n \l_@@_tmpc_tl
7818   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7819   \@@_qpoint:n \l_tmpa_tl
7820   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7821   \@@_qpoint:n { #1 }
7822   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7823   {
7824     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7825     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7826     \pgfusepath{stroke}
7827   }
7828   {
7829     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7830     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7831   }
7832 }

```

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).

```

7833 \cs_new_protected:Npn \@@_stroke_horizontal:n #1

```

```

7834 {
7835   \@@_qpoint:n \l_@@_tmpd_tl
7836   \clist_if_in:NnTF \l_@@_borders_clist { left }
7837     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7838     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7839   \@@_qpoint:n \l_tmpb_tl
7840   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7841   \@@_qpoint:n { #1 }
7842   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7843     {
7844       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7845       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7846       \pgfusepathqstroke
7847     }
7848     {
7849       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7850       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7851     }
7852 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7853 \keys_define:nn { NiceMatrix / BlockBorders }
7854 {
7855   borders .clist_set:N = \l_@@_borders_clist ,
7856   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7857   rounded-corners .default:n = 4 pt ,
7858   line-width .dim_set:N = \l_@@_line_width_dim
7859 }

```

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.

```

7860 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7861 {
7862   \begin { tikzpicture }
7863   \@@_clip_with_rounded_corners:
7864   \clist_map_inline:nn { #5 }
7865   {
7866     \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7867     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7868     (
7869       [
7870         xshift = \dim_use:N \l_@@_offset_dim ,
7871         yshift = - \dim_use:N \l_@@_offset_dim
7872       ]
7873       #1 -| #2
7874     )
7875     rectangle
7876     (
7877       [
7878         xshift = - \dim_use:N \l_@@_offset_dim ,
7879         yshift = \dim_use:N \l_@@_offset_dim
7880       ]
7881       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7882     ) ;
7883   }
7884   \end { tikzpicture }
7885 }
7886 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

```

```

7887 \keys_define:nn { NiceMatrix / SpecialOffset }
7888 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7889 \cs_set_protected:Npn \@@_renew_matrix:
7890 {
7891   \RenewDocumentEnvironment { pmatrix } { } {
7892     { \pNiceMatrix }
7893     { \endpNiceMatrix }
7894   \RenewDocumentEnvironment { vmatrix } { } {
7895     { \vNiceMatrix }
7896     { \endvNiceMatrix }
7897   \RenewDocumentEnvironment { Vmatrix } { } {
7898     { \VNiceMatrix }
7899     { \endVNiceMatrix }
7900   \RenewDocumentEnvironment { bmatrix } { } {
7901     { \bNiceMatrix }
7902     { \endbNiceMatrix }
7903   \RenewDocumentEnvironment { Bmatrix } { } {
7904     { \BNiceMatrix }
7905     { \endBNiceMatrix }
7906 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7907 \keys_define:nn { NiceMatrix / Auto }
7908 {
7909   columns-type .tl_set:N = \l_@@_columns_type_tl ,
7910   columns-type .value_required:n = true ,
7911   l .meta:n = { columns-type = l } ,
7912   r .meta:n = { columns-type = r } ,
7913   c .meta:n = { columns-type = c } ,
7914   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7915   delimiters / color .value_required:n = true ,
7916   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7917   delimiters / max-width .default:n = true ,
7918   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7919   delimiters .value_required:n = true ,
7920   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7921   rounded-corners .default:n = 4 pt
7922 }
7923 \NewDocumentCommand \AutoNiceMatrixWithDelims
7924 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7925 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7926 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
7927 {

```

The group is for the protection of the keys.

```

7928   \group_begin:
7929   \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl

```

We nullify the command `\@@_transform_preamble_i:` because we will provide a preamble which is yet transformed (by using `\l_@@_columns_type_tl` which is yet `nicematrix-ready`).

```

7930   % \bool_set_false:N \l_@@_preamble_bool
7931   \use:e
7932   {

```

```

7933 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7934 { * { #4 } { \exp_not:V \l_@@_columns_type_tl } }
7935 [ \exp_not:V \l_tmpa_tl ]
7936 }
7937 \int_if_zero:nT \l_@@_first_row_int
7938 {
7939 \int_if_zero:nT \l_@@_first_col_int { & }
7940 \prg_replicate:nn { #4 - 1 } { & }
7941 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7942 }
7943 \prg_replicate:nn { #3 }
7944 {
7945 \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`).

```

7946 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7947 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7948 }
7949 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7950 {
7951 \int_if_zero:nT \l_@@_first_col_int { & }
7952 \prg_replicate:nn { #4 - 1 } { & }
7953 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7954 }
7955 \end { NiceArrayWithDelims }
7956 \group_end:
7957 }

7958 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7959 {
7960 \cs_set_protected:cpn { #1 AutoNiceMatrix }
7961 {
7962 \bool_gset_true:N \g_@@_delims_bool
7963 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7964 \AutoNiceMatrixWithDelims { #2 } { #3 }
7965 }
7966 }

7967 \@@_define_com:nnn p ( )
7968 \@@_define_com:nnn b [ ]
7969 \@@_define_com:nnn v | |
7970 \@@_define_com:nnn V \ | \ |
7971 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7972 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
7973 {
7974 \group_begin:
7975 \bool_gset_false:N \g_@@_delims_bool
7976 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7977 \group_end:
7978 }

```

30 The redefinition of the command `\dotfill`

```

7979 \cs_set_eq:NN \@@_old_dotfill \dotfill
7980 \cs_new_protected:Npn \@@_dotfill:
7981 {

```

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}}`).

```

7982 \@@_old_dotfill
7983 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7984 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

7985 \cs_new_protected:Npn \@@_dotfill_i:
7986 { \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.

```

7987 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7988 {
7989 \tl_gput_right:Nx \g_@@_pre_code_after_tl
7990 {
7991 \@@_actually_diagbox:nnnnnn
7992 { \int_use:N \c@iRow }
7993 { \int_use:N \c@jCol }
7994 { \int_use:N \c@iRow }
7995 { \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.

```

7996 { \g_@@_row_style_tl \exp_not:n { #1 } }
7997 { \g_@@_row_style_tl \exp_not:n { #2 } }
7998 }

```

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`.

```

7999 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8000 {
8001 { \int_use:N \c@iRow }
8002 { \int_use:N \c@jCol }
8003 { \int_use:N \c@iRow }
8004 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8005 { }
8006 }
8007 }

```

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.

```

8008 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8009 {
8010 \pgfpicture
8011 \pgf@relevantforpicturesizefalse
8012 \pgfrememberpicturepositiononpagetrue
8013 \@@_qpoint:n { row - #1 }
8014 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8015 \@@_qpoint:n { col - #2 }

```



```

8016 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8017 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8018 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8019 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8020 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8021 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8022 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8023 {

```

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.

```

8024 \CT@arc@
8025 \pgfsetroundcap
8026 \pgfusepathqstroke
8027 }
8028 \pgfset { inner~sep = 1 pt }
8029 \pgfscope
8030 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8031 \pgfnode { rectangle } { south~west }
8032 {
8033 \begin { minipage } { 20 cm }
8034 \@@_math_toggle_token: #5 \@@_math_toggle_token:
8035 \end { minipage }
8036 }
8037 { }
8038 { }
8039 \endpgfscope
8040 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8041 \pgfnode { rectangle } { north~east }
8042 {
8043 \begin { minipage } { 20 cm }
8044 \raggedleft
8045 \@@_math_toggle_token: #6 \@@_math_toggle_token:
8046 \end { minipage }
8047 }
8048 { }
8049 { }
8050 \endpgfpicture
8051 }

```

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. 82.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8052 \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 `\`.

```

8053 \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`.

```

8054 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8055 {

```

```

8056 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8057 \@@_CodeAfter_iv:n
8058 }

```

We catch the argument of the command `\end` (in `#1`).

```

8059 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8060 {

```

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.

```

8061 \str_if_eq:eeTF \currenvir { #1 }
8062 { \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`.

```

8063 {
8064 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8065 \@@_CodeAfter_ii:n
8066 }
8067 }

```

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 columnn. 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.

```

8068 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8069 {
8070 \pgfpicture
8071 \pgfrememberpicturepositiononpagetrue
8072 \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.

```

8073 \@@_qpoint:n { row - 1 }
8074 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8075 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8076 \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).

```

8077 \bool_if:nTF { #3 }
8078 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8079 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8080 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8081 {
8082 \cs_if_exist:cT
8083 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8084 {
8085 \pgfpointanchor
8086 { \@@_env: - ##1 - #2 }
8087 { \bool_if:nTF { #3 } { west } { east } }

```

```

8088         \dim_set:Nn \l_tmpa_dim
8089         { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8090     }
8091 }

```

Now we can put the delimiter with a node of PGF.

```

8092     \pgfset { inner~sep = \c_zero_dim }
8093     \dim_zero:N \nulldelimiterspace
8094     \pgftransformshift
8095     {
8096         \pgfpoint
8097         { \l_tmpa_dim }
8098         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8099     }
8100     \pgfnode
8101     { rectangle }
8102     { \bool_if:nTF { #3 } { east } { west } }
8103     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8104         \nullfont
8105         \c_math_toggle_token
8106         \@@_color:V \l_@@_delimiters_color_tl
8107         \bool_if:nTF { #3 } { \left #1 } { \left . }
8108         \vcenter
8109         {
8110             \nullfont
8111             \hrule \@height
8112                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8113                 \@depth \c_zero_dim
8114                 \@width \c_zero_dim
8115         }
8116         \bool_if:nTF { #3 } { \right . } { \right #1 }
8117         \c_math_toggle_token
8118     }
8119     { }
8120     { }
8121 \endpgfpicture
8122 }

```

34 The command \SubMatrix

```

8123 \keys_define:nn { NiceMatrix / sub-matrix }
8124 {
8125     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8126     extra-height .value_required:n = true ,
8127     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8128     left-xshift .value_required:n = true ,
8129     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8130     right-xshift .value_required:n = true ,
8131     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8132     xshift .value_required:n = true ,
8133     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8134     delimiters / color .value_required:n = true ,
8135     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8136     slim .default:n = true ,
8137     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8138     hlines .default:n = all ,
8139     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8140     vlines .default:n = all ,
8141     hvlines .meta:n = { hlines, vlines } ,

```

```

8142     hvlines .value_forbidden:n = true
8143   }
8144   \keys_define:nn { NiceMatrix }
8145   {
8146     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8147     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8148     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8149     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8150   }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8151 \keys_define:nn { NiceMatrix / SubMatrix }
8152 {
8153   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8154   delimiters / color .value_required:n = true ,
8155   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8156   hlines .default:n = all ,
8157   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8158   vlines .default:n = all ,
8159   hvlines .meta:n = { hlines, vlines } ,
8160   hvlines .value_forbidden:n = true ,
8161   name .code:n =
8162     \tl_if_empty:nTF { #1 }
8163     { \@@_error:n { Invalid-name } }
8164     {
8165       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8166       {
8167         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8168         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8169         {
8170           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8171           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8172         }
8173       }
8174       { \@@_error:n { Invalid-name } }
8175     } ,
8176   name .value_required:n = true ,
8177   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8178   rules .value_required:n = true ,
8179   code .tl_set:N = \l_@@_code_tl ,
8180   code .value_required:n = true ,
8181   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8182 }

8183 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8184 {
8185   \peek_remove_spaces:n
8186   {
8187     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8188     {
8189       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8190       [
8191         delimiters / color = \l_@@_delimiters_color_tl ,
8192         hlines = \l_@@_submatrix_hlines_clist ,
8193         vlines = \l_@@_submatrix_vlines_clist ,
8194         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8195         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8196         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8197         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8198         #5
8199       ]
8200     }

```

```

8201 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8202 }
8203 }
8204 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8205 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8206 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8207 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8208 {
8209 \seq_gput_right:Nx \g_@@_submatrix_seq
8210 {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8211 { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8212 { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8213 { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8214 { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8215 }
8216 }

```

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`.

```

8217 \hook_gput_code:nnn { begindocument } { . }
8218 {
8219 \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8220 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8221 \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8222 {
8223 \peek_remove_spaces:n
8224 {
8225 \@@_sub_matrix:nnnnnnn
8226 { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8227 }
8228 }
8229 }

```

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).

```

8230 \NewDocumentCommand \@@_compute_i_j:nn
8231 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8232 { \@@_compute_i_j:nnnn #1 #2 }
8233 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8234 {
8235 \tl_set:Nn \l_@@_first_i_tl { #1 }
8236 \tl_set:Nn \l_@@_first_j_tl { #2 }
8237 \tl_set:Nn \l_@@_last_i_tl { #3 }
8238 \tl_set:Nn \l_@@_last_j_tl { #4 }

```

```

8239 \tl_if_eq:NnT \l_@@_first_i_tl { last }
8240 { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8241 \tl_if_eq:NnT \l_@@_first_j_tl { last }
8242 { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8243 \tl_if_eq:NnT \l_@@_last_i_tl { last }
8244 { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8245 \tl_if_eq:NnT \l_@@_last_j_tl { last }
8246 { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8247 }
8248 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8249 {
8250 \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8251 \@@_compute_i_j:nn { #2 } { #3 }
8252 % added 6.19b
8253 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8254 { \cs_set:Npn \arraystretch { 1 } }
8255 \bool_lazy_or:nnTF
8256 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8257 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8258 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8259 {
8260 \str_clear_new:N \l_@@_submatrix_name_str
8261 \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8262 \pgfpicture
8263 \pgfrememberpicturepositiononpagetrue
8264 \pgf@relevantforpicturesizefalse
8265 \pgfset { inner~sep = \c_zero_dim }
8266 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8267 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

8268 \bool_if:NTF \l_@@_submatrix_slim_bool
8269 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8270 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8271 {
8272 \cs_if_exist:cT
8273 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8274 {
8275 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8276 \dim_set:Nn \l_@@_x_initial_dim
8277 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8278 }
8279 \cs_if_exist:cT
8280 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8281 {
8282 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8283 \dim_set:Nn \l_@@_x_final_dim
8284 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8285 }
8286 }
8287 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8288 { \@@_error:nn { Impossible-delimiter } { left } }
8289 {
8290 \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8291 { \@@_error:nn { Impossible-delimiter } { right } }
8292 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8293 }
8294 \endpgfpicture
8295 }
8296 \group_end:
8297 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8298 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8299 {
8300   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8301   \dim_set:Nn \l_@@_y_initial_dim
8302   {
8303     \fp_to_dim:n
8304     {
8305       \pgf@y
8306       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8307     }
8308     } % modified 6.13c
8309   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8310   \dim_set:Nn \l_@@_y_final_dim
8311   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8312   % modified 6.13c
8313   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8314   {
8315     \cs_if_exist:cT
8316     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8317     {
8318       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8319       \dim_set:Nn \l_@@_y_initial_dim
8320       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8321     }
8322     \cs_if_exist:cT
8323     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8324     {
8325       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8326       \dim_set:Nn \l_@@_y_final_dim
8327       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8328     }
8329   }
8330   \dim_set:Nn \l_tmpa_dim
8331   {
8332     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8333     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8334   }
8335   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8336   \group_begin:
8337   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8338   \@@_set_CT@arc@:V \l_@@_rules_color_tl
8339   \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`.

```

8340   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8341   {
8342     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8343     {
8344       \int_compare:nNnT
8345       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8346       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8347       \@@_qpoint:n { col - ##1 }
8348       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8349       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8350       \pgfusepathqstroke
8351     }

```

```

8352     }
8353 }

```

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.

```

8354 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
8355 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8356 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8357 {
8358   \bool_lazy_and:nnTF
8359   { \int_compare_p:nNn { ##1 } > 0 }
8360   {
8361     \int_compare_p:nNn
8362     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8363   {
8364     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8365     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8366     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8367     \pgfusepathqstroke
8368   }
8369   { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
8370 }

```

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.

```

8371 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
8372 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8373 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8374 {
8375   \bool_lazy_and:nnTF
8376   { \int_compare_p:nNn { ##1 } > 0 }
8377   {
8378     \int_compare_p:nNn
8379     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8380   {
8381     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8382   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8383   \dim_set:Nn \l_tmpa_dim
8384   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8385   \str_case:nn { #1 }
8386   {
8387     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8388     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8389     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8390     }
8391     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8392   \dim_set:Nn \l_tmpb_dim
8393   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8394   \str_case:nn { #2 }
8395   {
8396     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8397     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8398     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8399   }
8400   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8401   \pgfusepathqstroke
8402   \group_end:
8403 }

```



```

8404         { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
8405     }

```

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).

```

8406     \str_if_empty:NF \l_@@_submatrix_name_str
8407     {
8408         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8409         \l_@@_x_initial_dim \l_@@_y_initial_dim
8410         \l_@@_x_final_dim \l_@@_y_final_dim
8411     }
8412     \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`.

```

8413     \begin { pgfscope }
8414     \pgftransformshift
8415     {
8416         \pgfpoint
8417         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8418         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8419     }
8420     \str_if_empty:NTF \l_@@_submatrix_name_str
8421     { \@@_node_left:nn #1 { } }
8422     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8423     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8424     \pgftransformshift
8425     {
8426         \pgfpoint
8427         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8428         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8429     }
8430     \str_if_empty:NTF \l_@@_submatrix_name_str
8431     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8432     {
8433         \@@_node_right:nnnn #2
8434         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8435     }
8436     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8437     \flag_clear_new:n { nicematrix }
8438     \l_@@_code_tl
8439 }

```

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`.

```

8440 \cs_set_eq:NN \@@_old_pgfpntanchor \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.

```

8441 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8442 {
8443     \use:e
8444     { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpointanchor_i:nn #1 } }
8445 }

```

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 -.

```
8446 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8447 { #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.

```
8448 \tl_const:Nn \c_@@_integers_alist_tl
8449 {
8450   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8451   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8452   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8453   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8454 }
```

```
8455 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8456 {
```

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`.

```
8457   \tl_if_empty:nTF { #2 }
8458   {
8459     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8460     {
8461       \flag_raise:n { nicematrix }
8462       \int_if_even:nTF { \flag_height:n { nicematrix } }
8463       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8464       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8465     }
8466     { #1 }
8467   }
```

If there is an hyphen, we have to see whether we have a node of the form i - j , row- i or col- j .

```
8468   { \@@_pgfpointanchor_iii:w { #1 } #2 }
8469 }
```

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`).

```
8470 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8471 {
8472   \str_case:nnF { #1 }
8473   {
8474     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8475     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8476   }
```

Now the case of a node of the form i - j .

```
8477   {
8478     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8479     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8480   }
8481 }
```

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`).

```

8482 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8483 {
8484   \pgfnode
8485     { rectangle }
8486     { east }
8487     {
8488       \nullfont
8489       \c_math_toggle_token
8490       \@@_color:V \l_@@_delimiters_color_tl
8491       \left #1
8492       \vcenter
8493         {
8494           \nullfont
8495           \hrule \@height \l_tmpa_dim
8496             \@depth \c_zero_dim
8497             \@width \c_zero_dim
8498         }
8499       \right .
8500       \c_math_toggle_token
8501     }
8502     { #2 }
8503     { }
8504 }

```

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.

```

8505 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8506 {
8507   \pgfnode
8508     { rectangle }
8509     { west }
8510     {
8511       \nullfont
8512       \c_math_toggle_token
8513       \@@_color:V \l_@@_delimiters_color_tl
8514       \left .
8515       \vcenter
8516         {
8517           \nullfont
8518           \hrule \@height \l_tmpa_dim
8519             \@depth \c_zero_dim
8520             \@width \c_zero_dim
8521         }
8522       \right #1
8523       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8524       ^ { \smash { #4 } }
8525       \c_math_toggle_token
8526     }
8527     { #2 }
8528     { }
8529 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8530 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8531 {
8532   \peek_remove_spaces:n
8533   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8534 }
8535 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8536 {
8537   \peek_remove_spaces:n
8538   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8539 }
8540 \keys_define:nn { NiceMatrix / Brace }
8541 {
8542   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8543   left-shorten .default:n = true ,
8544   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8545   shorten .meta:n = { left-shorten , right-shorten } ,
8546   right-shorten .default:n = true ,
8547   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8548   yshift .value_required:n = true ,
8549   yshift .initial:n = \c_zero_dim ,
8550   color .tl_set:N = \l_tmpa_tl ,
8551   color .value_required:n = true ,
8552   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8553 }

```

#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*.

```

8554 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8555 {
8556   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8557 \@@_compute_i_j:nn { #1 } { #2 }
8558 \bool_lazy_or:nnTF
8559 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8560 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8561 {
8562   \str_if_eq:nnTF { #5 } { under }
8563   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8564   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8565 }
8566 {
8567   \tl_clear:N \l_tmpa_tl
8568   \keys_set:nn { NiceMatrix / Brace } { #4 }
8569   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8570   \pgfpicture
8571   \pgfrememberpicturepositiononpagetrue
8572   \pgf@relevantforpicturesizefalse
8573   \bool_if:NT \l_@@_brace_left_shorten_bool
8574   {
8575     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8576     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8577     {
8578       \cs_if_exist:cT
8579       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8580       {
8581         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8582         \dim_set:Nn \l_@@_x_initial_dim
8583         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8584       }
8585     }

```

```

8586     }
8587 \bool_lazy_or:nnT
8588 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8589 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8590 {
8591   \@@_qpoint:n { col - \l_@@_first_j_tl }
8592   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8593 }
8594 \bool_if:NT \l_@@_brace_right_shorten_bool
8595 {
8596   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8597   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8598   {
8599     \cs_if_exist:cT
8600     { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8601     {
8602       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8603       \dim_set:Nn \l_@@_x_final_dim
8604       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8605     }
8606   }
8607 }
8608 \bool_lazy_or:nnT
8609 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8610 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8611 {
8612   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8613   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8614 }
8615 \pgfset { inner~sep = \c_zero_dim }
8616 \str_if_eq:nnTF { #5 } { under }
8617 { \@@_underbrace_i:n { #3 } }
8618 { \@@_overbrace_i:n { #3 } }
8619 \endpgfpicture
8620 }
8621 \group_end:
8622 }

```

The argument is the text to put above the brace.

```

8623 \cs_new_protected:Npn \@@_overbrace_i:n #1
8624 {
8625   \@@_qpoint:n { row - \l_@@_first_i_tl }
8626   \pgftransformshift
8627   {
8628     \pgfpoint
8629     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8630     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8631   }
8632   \pgfnode
8633   { rectangle }
8634   { south }
8635   {
8636     \vbox_top:n
8637     {
8638       \group_begin:
8639       \everycr { }
8640       \halign
8641       {
8642         \hfil ## \hfil \crrc
8643         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8644         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8645         \c_math_toggle_token
8646         \overbrace
8647         {

```

```

8648             \hbox_to_wd:nn
8649             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8650             { }
8651         }
8652         \c_math_toggle_token
8653     \cr
8654 }
8655 \group_end:
8656 }
8657 }
8658 { }
8659 { }
8660 }

```

The argument is the text to put under the brace.

```

8661 \cs_new_protected:Npn \@@_underbrace_i:n #1
8662 {
8663     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8664     \pgftransformshift
8665     {
8666         \pgfpoint
8667         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8668         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8669     }
8670     \pgfnode
8671     { rectangle }
8672     { north }
8673     {
8674         \group_begin:
8675         \everycr { }
8676         \vbox:n
8677         {
8678             \halign
8679             {
8680                 \hfil ## \hfil \crrc
8681                 \c_math_toggle_token
8682                 \underbrace
8683                 {
8684                     \hbox_to_wd:nn
8685                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8686                     { }
8687                 }
8688                 \c_math_toggle_token
8689                 \cr
8690                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8691                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8692             }
8693         }
8694         \group_end:
8695     }
8696     { }
8697     { }
8698 }

```

36 The command TikzEveryCell

```

8699 \bool_new:N \l_@@_not_empty_bool
8700 \bool_new:N \l_@@_empty_bool
8701

```

```

8702 \keys_define:nn { NiceMatrix / TikzEveryCell }
8703 {
8704   not-empty .code:n =
8705     \bool_lazy_or:nnTF
8706       \l_@@_in_code_after_bool
8707       \g_@@_recreate_cell_nodes_bool
8708       { \bool_set_true:N \l_@@_not_empty_bool }
8709       { \@@_error:n { detection-of-empty-cells } } ,
8710   not-empty .value_forbidden:n = true ,
8711   empty .code:n =
8712     \bool_lazy_or:nnTF
8713       \l_@@_in_code_after_bool
8714       \g_@@_recreate_cell_nodes_bool
8715       { \bool_set_true:N \l_@@_empty_bool }
8716       { \@@_error:n { detection-of-empty-cells } } ,
8717   empty .value_forbidden:n = true ,
8718   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
8719 }
8720
8721
8722 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8723 {
8724   \IfPackageLoadedTF { tikz }
8725   {
8726     \group_begin:
8727     \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.

```

8728     \tl_set:Nn \l_tmpa_tl { { #2 } }
8729     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8730       { \@@_for_a_block:nnnnn ##1 }
8731     \@@_all_the_cells:
8732     \group_end:
8733   }
8734   { \@@_error:n { TikzEveryCell-without-tikz } }
8735 }
8736
8737 \tl_new:N \@@_i_tl
8738 \tl_new:N \@@_j_tl
8739
8740 \cs_new_protected:Nn \@@_all_the_cells:
8741 {
8742   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8743   {
8744     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8745     {
8746       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8747       {
8748         \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8749           { \@@_i_tl - \@@_j_tl }
8750         {
8751           \bool_set_false:N \l_tmpa_bool
8752           \cs_if_exist:cTF
8753             { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8754             {
8755               \bool_if:NF \l_@@_empty_bool
8756               { \bool_set_true:N \l_tmpa_bool }
8757             }
8758             {
8759               \bool_if:NF \l_@@_not_empty_bool
8760               { \bool_set_true:N \l_tmpa_bool }
8761             }
8762           \bool_if:NT \l_tmpa_bool

```

```

8763         {
8764             \@@_block_tikz:nnnnV
8765             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8766         }
8767     }
8768 }
8769 }
8770 }
8771 }
8772
8773 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8774 {
8775     \bool_if:NF \l_@@_empty_bool
8776     {
8777         \@@_block_tikz:nnnnV
8778         { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8779     }
8780     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8781 }
8782
8783 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8784 {
8785     \int_step_inline:nnn { #1 } { #3 }
8786     {
8787         \int_step_inline:nnn { #2 } { #4 }
8788         { \cs_set:cpn { cell - ##1 - #####1 } { } }
8789     }
8790 }

```

37 The command \ShowCellNames

```

8791 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8792 {
8793     \dim_zero_new:N \g_@@_tmpc_dim
8794     \dim_zero_new:N \g_@@_tmpd_dim
8795     \dim_zero_new:N \g_@@_tmpe_dim
8796     \int_step_inline:nn \c@iRow
8797     {
8798         \begin { pgfpicture }
8799         \@@_qpoint:n { row - ##1 }
8800         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8801         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8802         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8803         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8804         \bool_if:NTF \l_@@_in_code_after_bool
8805         \end { pgfpicture }
8806         \int_step_inline:nn \c@jCol
8807         {
8808             \hbox_set:Nn \l_tmpa_box
8809             { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8810             \begin { pgfpicture }
8811             \@@_qpoint:n { col - #####1 }
8812             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8813             \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8814             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8815             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8816             \endpgfpicture
8817             \end { pgfpicture }
8818             \fp_set:Nn \l_tmpa_fp
8819             {
8820                 \fp_min:nn
8821                 {

```



```

8822         \fp_min:nn
8823         {
8824             \dim_ratio:nn
8825             { \g_@@_tmpd_dim }
8826             { \box_wd:N \l_tmpa_box }
8827         }
8828         {
8829             \dim_ratio:nn
8830             { \g_tmpb_dim }
8831             { \box_ht_plus_dp:N \l_tmpa_box }
8832         }
8833     }
8834     { 1.0 }
8835 }
8836 \box_scale:Nnn \l_tmpa_box
8837 { \fp_use:N \l_tmpa_fp }
8838 { \fp_use:N \l_tmpa_fp }
8839 \pgfpicture
8840 \pgfrememberpicturepositiononpagetrue
8841 \pgf@relevantforpicturesizefalse
8842 \pgftransformshift
8843 {
8844     \pgfpoint
8845     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8846     { \dim_use:N \g_tmpa_dim }
8847 }
8848 \pgfnode
8849 { rectangle }
8850 { center }
8851 { \box_use:N \l_tmpa_box }
8852 { }
8853 { }
8854 \endpgfpicture
8855 }
8856 }
8857 }
8858 \NewDocumentCommand \@@_ShowCellNames { }
8859 {
8860     \bool_if:NT \l_@@_in_code_after_bool
8861     {
8862         \pgfpicture
8863         \pgfrememberpicturepositiononpagetrue
8864         \pgf@relevantforpicturesizefalse
8865         \pgfpathrectanglecorners
8866         { \@@_qpoint:n { 1 } }
8867         {
8868             \@@_qpoint:n
8869             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8870         }
8871         \pgfsetfillopacity { 0.75 }
8872         \pgfsetfillcolor { white }
8873         \pgfusepathqfill
8874         \endpgfpicture
8875     }
8876     \dim_zero_new:N \g_@@_tmpc_dim
8877     \dim_zero_new:N \g_@@_tmpd_dim
8878     \dim_zero_new:N \g_@@_tmpe_dim
8879     \int_step_inline:nn \c@iRow
8880     {
8881         \bool_if:NTF \l_@@_in_code_after_bool
8882         {
8883             \pgfpicture
8884             \pgfrememberpicturepositiononpagetrue

```

```

8885         \pgf@relevantforpicturesizefalse
8886     }
8887     { \begin { pgfpicture } }
8888     \@@_qpoint:n { row - ##1 }
8889     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8890     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8891     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8892     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8893     \bool_if:NTF \l_@@_in_code_after_bool
8894     { \endpgfpicture }
8895     { \end { pgfpicture } }
8896     \int_step_inline:nn \c@jCol
8897     {
8898         \hbox_set:Nn \l_tmpa_box
8899         {
8900             \normalfont \Large \sffamily \bfseries
8901             \bool_if:NTF \l_@@_in_code_after_bool
8902             { \color { red } }
8903             { \color { red ! 50 } }
8904             ##1 - ####1
8905         }
8906         \bool_if:NTF \l_@@_in_code_after_bool
8907         {
8908             \pgfpicture
8909             \pgfrememberpicturerepositiononpagetrue
8910             \pgf@relevantforpicturesizefalse
8911         }
8912         { \begin { pgfpicture } }
8913         \@@_qpoint:n { col - ####1 }
8914         \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8915         \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8916         \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8917         \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8918         \bool_if:NTF \l_@@_in_code_after_bool
8919         { \endpgfpicture }
8920         { \end { pgfpicture } }
8921         \fp_set:Nn \l_tmpa_fp
8922         {
8923             \fp_min:nn
8924             {
8925                 \fp_min:nn
8926                 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
8927                 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
8928             }
8929             { 1.0 }
8930         }
8931         \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8932         \pgfpicture
8933         \pgfrememberpicturerepositiononpagetrue
8934         \pgf@relevantforpicturesizefalse
8935         \pgftransformshift
8936         {
8937             \pgfpoint
8938             { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8939             { \dim_use:N \g_tmpa_dim }
8940         }
8941         \pgfnode
8942         { rectangle }
8943         { center }
8944         { \box_use:N \l_tmpa_box }
8945         { }
8946         { }
8947         \endpgfpicture

```

```

8948     }
8949   }
8950 }

```

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.

```

8951 \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.

```

8952 \bool_new:N \g_@@_footnote_bool

8953 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
8954 {
8955   The~key~'\l_keys_key_str'~is~unknown. \\
8956   That~key~will~be~ignored. \\
8957   For~a~list~of~the~available~keys,~type~H~<return>.
8958 }
8959 {
8960   The~available~keys~are~(in~alphabetic~order):~
8961   footnote,~
8962   footnotehyper,~
8963   messages~for~Overleaf,~
8964   no~test~for~array,~
8965   renew~dots,~and~
8966   renew~matrix.
8967 }

8968 \keys_define:nn { NiceMatrix / Package }
8969 {
8970   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
8971   renew~dots .value_forbidden:n = true ,
8972   renew~matrix .code:n = \@@_renew_matrix: ,
8973   renew~matrix .value_forbidden:n = true ,
8974   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
8975   footnote .bool_set:N = \g_@@_footnote_bool ,
8976   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
8977   no~test~for~array .bool_set:N = \g_@@_no_test_for_array_bool ,
8978   no~test~for~array .default:n = true ,
8979   unknown .code:n = \@@_error:n { Unknown~key~for~package }
8980 }
8981 \ProcessKeysOptions { NiceMatrix / Package }

8982 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8983 {
8984   You~can't~use~the~option~'footnote'~because~the~package~
8985   footnotehyper~has~already~been~loaded.~
8986   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8987   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8988   of~the~package~footnotehyper.\\
8989   The~package~footnote~won't~be~loaded.
8990 }

```

```

8991 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8992 {
8993   You~can't~use~the~option~'footnotehyper'~because~the~package~
8994   footnote~has~already~been~loaded.~
8995   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8996   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8997   of~the~package~footnote.\\
8998   The~package~footnotehyper~won't~be~loaded.
8999 }

9000 \bool_if:NT \g_@@_footnote_bool
9001 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9002   \IfClassLoadedTF { beamer }
9003   { \bool_set_false:N \g_@@_footnote_bool }
9004   {
9005     \IfPackageLoadedTF { footnotehyper }
9006     { \@@_error:n { footnote~with~footnotehyper~package } }
9007     { \usepackage { footnote } }
9008   }
9009 }

9010 \bool_if:NT \g_@@_footnotehyper_bool
9011 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9012   \IfClassLoadedTF { beamer }
9013   { \bool_set_false:N \g_@@_footnote_bool }
9014   {
9015     \IfPackageLoadedTF { footnote }
9016     { \@@_error:n { footnotehyper~with~footnote~package } }
9017     { \usepackage { footnotehyper } }
9018   }
9019   \bool_set_true:N \g_@@_footnote_bool
9020 }

```

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.

```

9021 \bool_new:N \l_@@_underscore_loaded_bool
9022 \IfPackageLoadedTF { underscore }
9023 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9024 { }

9025 \hook_gput_code:nnn { begindocument } { . }
9026 {
9027   \bool_if:NF \l_@@_underscore_loaded_bool
9028   {
9029     \IfPackageLoadedTF { underscore }
9030     { \@@_error:n { underscore~after~nicematrix } }
9031     { }
9032   }
9033 }

```

40 Error messages of the package

```

9034 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9035 { \str_const:Nn \c_@@_available_keys_str { } }
9036 {
9037   \str_const:Nn \c_@@_available_keys_str
9038   { For~a~list~of~the~available~keys,~type~H~<return>. }
9039 }

9040 \seq_new:N \g_@@_types_of_matrix_seq
9041 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9042 {
9043   NiceMatrix ,
9044   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9045 }
9046 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9047 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9048 \cs_new_protected:Npn \@@_error_too_much_cols:
9049 {
9050   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9051   {
9052     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9053     { \@@_fatal:n { too~much~cols~for~matrix } }
9054     {
9055       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9056       { \@@_fatal:n { too~much~cols~for~matrix } }
9057       {
9058         \bool_if:NF \l_@@_last_col_without_value_bool
9059         { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9060       }
9061     }
9062   }
9063   { \@@_fatal:nn { too~much~cols~for~array } }
9064 }

```

The following command must *not* be protected since it's used in an error message.

```

9065 \cs_new:Npn \@@_message_hdotsfor:
9066 {
9067   \tl_if_empty:VF \g_@@_HVDotsfor_lines_tl
9068   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9069 }

9070 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9071 {
9072   Incompatible~options.\\
9073   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9074   The~output~will~not~be~reliable.
9075 }

9076 \@@_msg_new:nn { negative~weight }
9077 {
9078   Negative~weight.\\
9079   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9080   the~value~'\int_use:N \l_@@_weight_int'.\\
9081   The~absolute~value~will~be~used.
9082 }

9083 \@@_msg_new:nn { last~col~not~used }
9084 {
9085   Column~not~used.\\

```

```

9086     The-key~'last-col'~is-in-force-but~you-have-not-used-that-last-column~
9087     in-your~\@@_full_name_env:~.~However,~you~can~go-on.
9088 }

9089 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9090 {
9091     Too-much-columns.\\
9092     In-the-row~\int_eval:n { \c@iRow },~
9093     you-try-to-use-more-columns~
9094     than-allowed-by-your~\@@_full_name_env:~\@@_message_hdotsfor:~\
9095     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9096     (plus-the-exterior-columns).~This-error-is-fatal.
9097 }

9098 \@@_msg_new:nn { too-much-cols-for-matrix }
9099 {
9100     Too-much-columns.\\
9101     In-the-row~\int_eval:n { \c@iRow },~
9102     you-try-to-use-more-columns-than-allowed-by-your~
9103     \@@_full_name_env:~\@@_message_hdotsfor:~\ Recall~that~the~maximal~
9104     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9105     columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9106     Its~current~value~is~\int_use:N \c@MaxMatrixCols~(use~
9107     \token_to_str:N \setcounter~ to~change~that~value).~
9108     This-error-is-fatal.
9109 }

9110 \@@_msg_new:nn { too-much-cols-for-array }
9111 {
9112     Too-much-columns.\\
9113     In-the-row~\int_eval:n { \c@iRow },~
9114     ~you-try-to-use-more-columns-than-allowed-by-your~
9115     \@@_full_name_env:~\@@_message_hdotsfor:~\ The~maximal~number-of~columns~is~
9116     \int_use:N \g_@@_static_num_of_col_int~
9117     ~(plus~the~potential~exterior~ones).
9118     This-error-is-fatal.
9119 }

9120 \@@_msg_new:nn { columns-not-used }
9121 {
9122     Columns-not-used.\\
9123     The-preamble-of-your~\@@_full_name_env:~\ announces~\int_use:N
9124     \g_@@_static_num_of_col_int~ columns-but-you-use-only~\int_use:N \c@jCol.\\
9125     The-columns-you-did-not-used-won't-be-created.\\
9126     You-won't-have-similar-error-till-the-end-of-the-document.
9127 }

9128 \@@_msg_new:nn { in-first-col }
9129 {
9130     Erroneous-use.\\
9131     You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9132     That-command-will-be-ignored.
9133 }

9134 \@@_msg_new:nn { in-last-col }
9135 {
9136     Erroneous-use.\\
9137     You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9138     That-command-will-be-ignored.
9139 }

9140 \@@_msg_new:nn { in-first-row }
9141 {
9142     Erroneous-use.\\
9143     You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9144     That-command-will-be-ignored.
9145 }

```

```

9146 \@@_msg_new:nn { in~last~row }
9147 {
9148   You~can't~use~the~command~\#1 in~the~last~row~(exterior)~of~the~array.\\
9149   That~command~will~be~ignored.
9150 }
9151 \@@_msg_new:nn { caption~outside~float }
9152 {
9153   Key~caption~forbidden.\\
9154   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9155   environment.~This~key~will~be~ignored.
9156 }
9157 \@@_msg_new:nn { short~caption~without~caption }
9158 {
9159   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9160   However,~your~'short~caption'~will~be~used~as~'caption'.
9161 }
9162 \@@_msg_new:nn { double~closing~delimiter }
9163 {
9164   Double~delimiter.\\
9165   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9166   delimiter.~This~delimiter~will~be~ignored.
9167 }
9168 \@@_msg_new:nn { delimiter~after~opening }
9169 {
9170   Double~delimiter.\\
9171   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9172   delimiter.~That~delimiter~will~be~ignored.
9173 }
9174 \@@_msg_new:nn { bad~option~for~line~style }
9175 {
9176   Bad~line~style.\\
9177   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9178   is~'standard'.~That~key~will~be~ignored.
9179 }
9180 \@@_msg_new:nn { Identical~notes~in~caption }
9181 {
9182   Identical~tabular~notes.\\
9183   You~can't~put~several~notes~with~the~same~content~in~
9184   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9185   If~you~go~on,~the~output~will~probably~be~erroneous.
9186 }
9187 \@@_msg_new:nn { tabularnote~below~the~tabular }
9188 {
9189   \token_to_str:N \tabularnote\ forbidden\\
9190   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9191   of~your~tabular~because~the~caption~will~be~composed~below~
9192   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9193   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9194   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9195   no~similar~error~will~raised~in~this~document.
9196 }
9197 \@@_msg_new:nn { Unknown~key~for~rules }
9198 {
9199   Unknown~key.\\
9200   There~is~only~two~keys~available~here:~width~and~color.\\
9201   Your~key~'\l_keys_key_str'~will~be~ignored.
9202 }
9203 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9204 {
9205   Unknown~key.\\

```

```

9206     There~is~only~two~keys~available~here:~
9207     'empty'~and~'not-empty'.\\
9208     Your~key~'\l_keys_key_str'~will~be~ignored.
9209 }

9210 \@@_msg_new:nn { Unknown~key~for~rotate }
9211 {
9212     Unknown~key.\\
9213     The~only~key~available~here~is~'c'.\\
9214     Your~key~'\l_keys_key_str'~will~be~ignored.
9215 }

9216 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9217 {
9218     Unknown~key.\\
9219     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9220     It~you~go~on,~you~will~probably~have~other~errors. \\
9221     \c_@@_available_keys_str
9222 }
9223 {
9224     The~available~keys~are~(in~alphabetic~order):~
9225     ccommand,~
9226     color,~
9227     command,~
9228     dotted,~
9229     letter,~
9230     multiplicity,~
9231     sep-color,~
9232     tikz,~and~total-width.
9233 }

9234 \@@_msg_new:nnn { Unknown~key~for~xdots }
9235 {
9236     Unknown~key.\\
9237     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9238     \c_@@_available_keys_str
9239 }
9240 {
9241     The~available~keys~are~(in~alphabetic~order):~
9242     'color',~
9243     'horizontal-labels',~
9244     'inter',~
9245     'line-style',~
9246     'radius',~
9247     'shorten',~
9248     'shorten-end'~and~'shorten-start'.
9249 }

9250 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9251 {
9252     Unknown~key.\\
9253     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9254     (and~you~try~to~use~'\l_keys_key_str')\\
9255     That~key~will~be~ignored.
9256 }

9257 \@@_msg_new:nn { label~without~caption }
9258 {
9259     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9260     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9261 }

9262 \@@_msg_new:nn { W~warning }
9263 {
9264     Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9265     (row~\int_use:N \c@iRow).
9266 }

```



```

9267 \@@_msg_new:nn { Construct-too-large }
9268 {
9269   Construct-too-large.\
9270   Your~command~\token_to_str:N #1
9271   can't~be~drawn~because~your~matrix~is~too~small.\
9272   That~command~will~be~ignored.
9273 }
9274 \@@_msg_new:nn { underscore~after~nicematrix }
9275 {
9276   Problem~with~'underscore'.\
9277   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9278   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\
9279   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9280 }
9281 \@@_msg_new:nn { ampersand~in~light-syntax }
9282 {
9283   Ampersand~forbidden.\
9284   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9285   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9286 }
9287 \@@_msg_new:nn { double-backslash~in~light-syntax }
9288 {
9289   Double~backslash~forbidden.\
9290   You~can't~use~\token_to_str:N
9291   \~to~separate~rows~because~the~key~'light-syntax'~
9292   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9293   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9294 }
9295 \@@_msg_new:nn { hlines~with~color }
9296 {
9297   Incompatible~keys.\
9298   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9299   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\
9300   Maybe~it~will~possible~in~future~version.\
9301   Your~key~will~be~discarded.
9302 }
9303 \@@_msg_new:nn { bad~value~for~baseline }
9304 {
9305   Bad~value~for~baseline.\
9306   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9307   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9308   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9309   the~form~'line-i'.\
9310   A~value~of~1~will~be~used.
9311 }
9312 \@@_msg_new:nn { detection~of~empty~cells }
9313 {
9314   Problem~with~'not-empty'\
9315   For~technical~reasons,~you~must~activate~
9316   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9317   in~order~to~use~the~key~'\l_keys_key_str'.\
9318   That~key~will~be~ignored.
9319 }
9320 \@@_msg_new:nn { siunitx~not~loaded }
9321 {
9322   siunitx~not~loaded\
9323   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\
9324   That~error~is~fatal.
9325 }
9326 \@@_msg_new:nn { ragged2e~not~loaded }

```

```

9327 {
9328     You-have-to-load~'ragged2e'~in-order-to-use-the-key~'\l_keys_key_str'~in~
9329     your~column~'\l_@@_vpos_col_str'~(or~'X')~.~The-key~'\str_lowercase:V
9330     \l_keys_key_str'~will-be-used-instead.
9331 }
9332 \@@_msg_new:nn { Invalid-name }
9333 {
9334     Invalid-name.\\
9335     You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
9336     \SubMatrix\ of~your~\@@_full_name_env:~.\\
9337     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
9338     This-key-will-be-ignored.
9339 }
9340 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9341 {
9342     Wrong-line.\\
9343     You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
9344     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but-that~
9345     number~is-not-valid.~It-will-be-ignored.
9346 }
9347 \@@_msg_new:nn { Impossible-delimiter }
9348 {
9349     Impossible-delimiter.\\
9350     It's-impossible-to-draw-the~#1~delimiter-of~your~
9351     \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
9352     in-that-column.
9353     \bool_if:NT \l_@@_submatrix_slim_bool
9354     { ~Maybe-you-should-try-without-the-key~'slim'. } \\
9355     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9356 }
9357 \@@_msg_new:nnn { width-without-X-columns }
9358 {
9359     You-have-used-the-key~'width'~but-you-have-put-no~'X'~column.~
9360     That-key-will-be-ignored.
9361 }
9362 {
9363     This-message-is-the-message~'width-without-X-columns'~
9364     of-the-module~'nicematrix'.~
9365     The-experimented-users-can-disable-that-message-with~
9366     \token_to_str:N \msg_redirect_name:nnn.\\
9367 }
9368
9369 \@@_msg_new:nn { key-multiplicity-with-dotted }
9370 {
9371     Incompatible-keys. \\
9372     You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
9373     in-a~'custom-line'~.~They-are-incompatible. \\
9374     The-key~'multiplicity'~will-be-discarded.
9375 }
9376 \@@_msg_new:nn { empty-environment }
9377 {
9378     Empty-environment.\\
9379     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9380 }
9381 \@@_msg_new:nn { No-letter-and-no-command }
9382 {
9383     Erroneous-use.\\
9384     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9385     key~'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys~'command'~or~
9386     ~'ccommand'~(to-draw-horizontal-rules).\\
9387     However,~you-can-go-on.

```

```

9388 }
9389 \@@_msg_new:nn { Forbidden-letter }
9390 {
9391   Forbidden-letter.\
9392   You-can't-use-the-letter~'#1'~for-a-customized-line.\
9393   It-will-be-ignored.
9394 }
9395 \@@_msg_new:nn { Several-letters }
9396 {
9397   Wrong-name.\
9398   You-must-use-only-one-letter-as-value-for-the-key~'letter'~(and-you-
9399   have-used~'\l_@@_letter_str').\
9400   It-will-be-ignored.
9401 }
9402 \@@_msg_new:nn { Delimiter-with-small }
9403 {
9404   Delimiter~forbidden.\
9405   You-can't-put-a-delimiter-in-the-preamble-of~your~\@@_full_name_env:\
9406   because-the-key~'small'~is-in-force.\
9407   This-error-is-fatal.
9408 }
9409 \@@_msg_new:nn { unknown-cell-for~line~in~CodeAfter }
9410 {
9411   Unknown-cell.\
9412   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9413   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9414   can't-be-executed-because-a-cell~doesn't-exist.\
9415   This-command~\token_to_str:N \line\ will-be-ignored.
9416 }
9417 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9418 {
9419   Duplicate-name.\
9420   The-name~'#1'~is-already-used-for-a~\token_to_str:N \SubMatrix\
9421   in~this~\@@_full_name_env:.\
9422   This-key-will-be-ignored.\
9423   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9424   { For-a-list-of-the-names-already-used,~type-H<return>. }
9425 }
9426 {
9427   The-names-already-defined-in~this~\@@_full_name_env:\ are:~
9428   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9429 }
9430 \@@_msg_new:nn { r-or-l-with-preamble }
9431 {
9432   Erroneous-use.\
9433   You-can't-use-the-key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9434   You-must-specify-the-alignment-of~your~columns~with~the~preamble~of~
9435   your~\@@_full_name_env:.\
9436   This-key-will-be-ignored.
9437 }
9438 \@@_msg_new:nn { Hdotsfor~in~col-0 }
9439 {
9440   Erroneous-use.\
9441   You-can't-use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9442   the~array.~This-error-is-fatal.
9443 }
9444 \@@_msg_new:nn { bad-corner }
9445 {
9446   Bad-corner.\
9447   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~

```

```

9448     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9449     This~specification~of~corner~will~be~ignored.
9450 }
9451 \@@_msg_new:nn { bad~border }
9452 {
9453     Bad~border.\\
9454     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9455     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9456     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9457     also~use~the~key~'tikz'
9458     \IfPackageLoadedTF { tikz }
9459     { }
9460     {~if~you~load~the~LaTeX~package~'tikz'}).\\
9461     This~specification~of~border~will~be~ignored.
9462 }
9463 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9464 {
9465     TikZ~not~loaded.\\
9466     You~can't~use~\token_to_str:N \TikzEveryCell\
9467     because~you~have~not~loaded~tikz.~
9468     This~command~will~be~ignored.
9469 }
9470 \@@_msg_new:nn { tikz~key~without~tikz }
9471 {
9472     TikZ~not~loaded.\\
9473     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9474     \Block'~because~you~have~not~loaded~tikz.~
9475     This~key~will~be~ignored.
9476 }
9477 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9478 {
9479     Erroneous~use.\\
9480     In~the~\@@_full_name_env:,~you~must~use~the~key~
9481     'last~col'~without~value.\\
9482     However,~you~can~go~on~for~this~time~
9483     (the~value~'\l_keys_value_tl'~will~be~ignored).
9484 }
9485 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
9486 {
9487     Erroneous~use.\\
9488     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9489     'last~col'~without~value.\\
9490     However,~you~can~go~on~for~this~time~
9491     (the~value~'\l_keys_value_tl'~will~be~ignored).
9492 }
9493 \@@_msg_new:nn { Block~too~large~1 }
9494 {
9495     Block~too~large.\\
9496     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9497     too~small~for~that~block. \\
9498     This~block~and~maybe~others~will~be~ignored.
9499 }
9500 \@@_msg_new:nn { Block~too~large~2 }
9501 {
9502     Block~too~large.\\
9503     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9504     \g_@@_static_num_of_col_int\
9505     columns~but~you~use~only~\int_use:N \c_jCol\ and~that's~why~a~block~
9506     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9507     (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:~\\
9508     This~block~and~maybe~others~will~be~ignored.

```

```

9509 }
9510 \@@_msg_new:nn { unknown~column~type }
9511 {
9512   Bad~column~type.\\
9513   The~column~type~'#1'~in~your~\@@_full_name_env:\
9514   is~unknown. \\
9515   This~error~is~fatal.
9516 }
9517 \@@_msg_new:nn { unknown~column~type~S }
9518 {
9519   Bad~column~type.\\
9520   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9521   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9522   load~that~package. \\
9523   This~error~is~fatal.
9524 }
9525 \@@_msg_new:nn { tabularnote~forbidden }
9526 {
9527   Forbidden~command.\\
9528   You~can't~use~the~command~\token_to_str:N\tabularnote\
9529   ~here.~This~command~is~available~only~in~
9530   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9531   the~argument~of~a~command~\token_to_str:N \caption\ included~
9532   in~an~environment~{table}. \\
9533   This~command~will~be~ignored.
9534 }
9535 \@@_msg_new:nn { borders~forbidden }
9536 {
9537   Forbidden~key.\\
9538   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9539   because~the~option~'rounded~corners'~
9540   is~in~force~with~a~non~zero~value.\\
9541   This~key~will~be~ignored.
9542 }
9543 \@@_msg_new:nn { bottomrule~without~booktabs }
9544 {
9545   booktabs~not~loaded.\\
9546   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9547   loaded~'booktabs'.\\
9548   This~key~will~be~ignored.
9549 }
9550 \@@_msg_new:nn { enumitem~not~loaded }
9551 {
9552   enumitem~not~loaded.\\
9553   You~can't~use~the~command~\token_to_str:N\tabularnote\
9554   ~because~you~haven't~loaded~'enumitem'.\\
9555   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9556   ignored~in~the~document.
9557 }
9558 \@@_msg_new:nn { tikz~without~tikz }
9559 {
9560   Tikz~not~loaded.\\
9561   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9562   loaded.~If~you~go~on,~that~key~will~be~ignored.
9563 }
9564 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9565 {
9566   Tikz~not~loaded.\\
9567   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9568   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~

```

```

9569     You-can-go-on-but-you-will-have-another-error-if-you-actually-
9570     use-that-custom-line.
9571 }

9572 \@@_msg_new:nn { tikz-in-borders-without-tikz }
9573 {
9574     Tikz-not-loaded.\\
9575     You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9576     command~'\token_to_str:N\Block')~but-tikz-is-not-loaded.~
9577     That-key-will-be-ignored.
9578 }

9579 \@@_msg_new:nn { without-color-inside }
9580 {
9581     If-order-to-use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9582     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9583     outside~\token_to_str:N \CodeBefore,~you~
9584     should-have-used-the-key~'color-inside'~in-your~\@@_full_name_env:~\\
9585     You-can-go-on-but-you-may-need-more-compilations.
9586 }

9587 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9588 {
9589     Erroneous-use.\\
9590     In-a~'custom-line',~you-have-used-both~'tikz'~and~'color',~
9591     which-is-forbidden~(you-should-use~'color'~inside~the-key~'tikz').~
9592     The-key~'color'~will-be-discarded.
9593 }

9594 \@@_msg_new:nn { Wrong-last-row }
9595 {
9596     Wrong-number.\\
9597     You-have-used~'last-row=\int_use:N \l_@@_last_row_int'~but-your~
9598     \@@_full_name_env:\ seems-to-have~\int_use:N \c@iRow \ rows.~
9599     If-you-go-on,~the-value-of~\int_use:N \c@iRow \ will-be-used-for~
9600     last-row.~You-can-avoid-this-problem-by-using~'last-row'~
9601     without-value~(more-compilations-might-be-necessary).
9602 }

9603 \@@_msg_new:nn { Yet-in-env }
9604 {
9605     Nested-environments.\\
9606     Environments-of~nicematrix~can't-be-nested.\\
9607     This-error-is-fatal.
9608 }

9609 \@@_msg_new:nn { Outside-math-mode }
9610 {
9611     Outside-math-mode.\\
9612     The~\@@_full_name_env:\ can-be-used-only-in-math-mode~
9613     (and-not-in~\token_to_str:N \vcenter).\\
9614     This-error-is-fatal.
9615 }

9616 \@@_msg_new:nn { One-letter-allowed }
9617 {
9618     Bad-name.\\
9619     The-value-of-key~'\l_keys_key_str'~must-be-of-length~1.\\
9620     It-will-be-ignored.
9621 }

9622 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9623 {
9624     Environment~{TabularNote}~forbidden.\\
9625     You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
9626     but~*before*~the~\token_to_str:N \CodeAfter.\\
9627     This-environment~{TabularNote}~will-be-ignored.
9628 }

```

```

9629 \@@_msg_new:nn { varwidth~not~loaded }
9630 {
9631   varwidth~not~loaded.\\
9632   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9633   loaded.\\
9634   Your~column~will~behave~like~'p'.
9635 }
9636 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
9637 {
9638   Unknown~key.\\
9639   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9640   \c_@@_available_keys_str
9641 }
9642 {
9643   The~available~keys~are~(in~alphabetic~order):~
9644   color,~
9645   dotted,~
9646   multiplicity,~
9647   sep~color,~
9648   tikz,~and~total~width.
9649 }
9650
9651 \@@_msg_new:nnn { Unknown~key~for~Block }
9652 {
9653   Unknown~key.\\
9654   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9655   \Block.\\ It~will~be~ignored. \\
9656   \c_@@_available_keys_str
9657 }
9658 {
9659   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9660   hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9661   respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9662 }
9663 \@@_msg_new:nnn { Unknown~key~for~Brace }
9664 {
9665   Unknown~key.\\
9666   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9667   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9668   It~will~be~ignored. \\
9669   \c_@@_available_keys_str
9670 }
9671 {
9672   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9673   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9674   right~shorten)~and~yshift.
9675 }
9676 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9677 {
9678   Unknown~key.\\
9679   The~key~'\l_keys_key_str'~is~unknown.\\
9680   It~will~be~ignored. \\
9681   \c_@@_available_keys_str
9682 }
9683 {
9684   The~available~keys~are~(in~alphabetic~order):~
9685   delimiters/color,~
9686   rules~(with~the~subkeys~'color'~and~'width'),~
9687   sub~matrix~(several~subkeys)~
9688   and~xdots~(several~subkeys).~
9689   The~latter~is~for~the~command~\token_to_str:N \line.
9690 }

```

```

9691 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9692 {
9693   Unknown~key.\\
9694   The~key~'\l_keys_key_str'~is~unknown.\\
9695   It~will~be~ignored. \\
9696   \c_@@_available_keys_str
9697 }
9698 {
9699   The~available~keys~are~(in~alphabetic~order):~
9700   create-cell-nodes,~
9701   delimiters/color~and~
9702   sub-matrix~(several~subkeys).
9703 }
9704 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9705 {
9706   Unknown~key.\\
9707   The~key~'\l_keys_key_str'~is~unknown.\\
9708   That~key~will~be~ignored. \\
9709   \c_@@_available_keys_str
9710 }
9711 {
9712   The~available~keys~are~(in~alphabetic~order):~
9713   'delimiters/color',~
9714   'extra-height',~
9715   'hlines',~
9716   'hvlines',~
9717   'left-xshift',~
9718   'name',~
9719   'right-xshift',~
9720   'rules'~(with~the~subkeys~'color'~and~'width'),~
9721   'slim',~
9722   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9723   and~'right-xshift').\\
9724 }
9725 \@@_msg_new:nnn { Unknown~key~for~notes }
9726 {
9727   Unknown~key.\\
9728   The~key~'\l_keys_key_str'~is~unknown.\\
9729   That~key~will~be~ignored. \\
9730   \c_@@_available_keys_str
9731 }
9732 {
9733   The~available~keys~are~(in~alphabetic~order):~
9734   bottomrule,~
9735   code-after,~
9736   code-before,~
9737   detect-duplicates,~
9738   enumitem-keys,~
9739   enumitem-keys-para,~
9740   para,~
9741   label-in-list,~
9742   label-in-tabular~and~
9743   style.
9744 }
9745 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9746 {
9747   Unknown~key.\\
9748   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9749   \token_to_str:N \RowStyle. \\
9750   That~key~will~be~ignored. \\
9751   \c_@@_available_keys_str
9752 }
9753 {

```



```

9754 The~available~keys~are~(in~alphabetic~order):~
9755 'bold',~
9756 'cell-space-top-limit',~
9757 'cell-space-bottom-limit',~
9758 'cell-space-limits',~
9759 'color',~
9760 'nb-rows'~and~
9761 'rowcolor'.
9762 }
9763 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9764 {
9765   Unknown~key.\\
9766   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9767   \token_to_str:N \NiceMatrixOptions. \\
9768   That~key~will~be~ignored. \\
9769   \c_@@_available_keys_str
9770 }
9771 {
9772   The~available~keys~are~(in~alphabetic~order):~
9773   allow-duplicate-names,~
9774   caption-above,~
9775   cell-space-bottom-limit,~
9776   cell-space-limits,~
9777   cell-space-top-limit,~
9778   code-for-first-col,~
9779   code-for-first-row,~
9780   code-for-last-col,~
9781   code-for-last-row,~
9782   corners,~
9783   custom-key,~
9784   create-extra-nodes,~
9785   create-medium-nodes,~
9786   create-large-nodes,~
9787   delimiters~(several~subkeys),~
9788   end-of-row,~
9789   first-col,~
9790   first-row,~
9791   hlines,~
9792   hvlines,~
9793   hvlines-except-borders,~
9794   last-col,~
9795   last-row,~
9796   left-margin,~
9797   light-syntax,~
9798   matrix/columns-type,~
9799   notes~(several~subkeys),~
9800   nullify-dots,~
9801   pgf-node-code,~
9802   renew-dots,~
9803   renew-matrix,~
9804   respect-arraystretch,~
9805   rounded-corners,~
9806   right-margin,~
9807   rules~(with~the~subkeys~'color'~and~'width'),~
9808   small,~
9809   sub-matrix~(several~subkeys),~
9810   vlimes,~
9811   xdots~(several~subkeys).
9812 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9813 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9814 {

```

```

9815     Unknown~key.\\
9816     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9817     \{NiceArray\}. \\
9818     That~key~will~be~ignored. \\
9819     \c_@@_available_keys_str
9820 }
9821 {
9822     The~available~keys~are~(in~alphabetic~order):~
9823     b,~
9824     baseline,~
9825     c,~
9826     cell-space-bottom-limit,~
9827     cell-space-limits,~
9828     cell-space-top-limit,~
9829     code-after,~
9830     code-for-first-col,~
9831     code-for-first-row,~
9832     code-for-last-col,~
9833     code-for-last-row,~
9834     color-inside,~
9835     columns-width,~
9836     corners,~
9837     create-extra-nodes,~
9838     create-medium-nodes,~
9839     create-large-nodes,~
9840     extra-left-margin,~
9841     extra-right-margin,~
9842     first-col,~
9843     first-row,~
9844     hlines,~
9845     hvlines,~
9846     hvlines-except-borders,~
9847     last-col,~
9848     last-row,~
9849     left-margin,~
9850     light-syntax,~
9851     name,~
9852     nullify-dots,~
9853     pgf-node-code,~
9854     renew-dots,~
9855     respect-arraystretch,~
9856     right-margin,~
9857     rounded-corners,~
9858     rules~(with~the~subkeys~'color'~and~'width'),~
9859     small,~
9860     t,~
9861     vlines,~
9862     xdots/color,~
9863     xdots/shorten-start,~
9864     xdots/shorten-end,~
9865     xdots/shorten-and~
9866     xdots/line-style.
9867 }

```

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).

```

9868 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9869 {
9870     Unknown~key.\\
9871     The~key~'\l_keys_key_str'~is~unknown~for~the~
9872     \@@_full_name_env:. \\
9873     That~key~will~be~ignored. \\
9874     \c_@@_available_keys_str
9875 }

```

```

9876 {
9877   The~available~keys~are~(in~alphabetic~order):~
9878   b,~
9879   baseline,~
9880   c,~
9881   cell-space-bottom-limit,~
9882   cell-space-limits,~
9883   cell-space-top-limit,~
9884   code-after,~
9885   code-for-first-col,~
9886   code-for-first-row,~
9887   code-for-last-col,~
9888   code-for-last-row,~
9889   color-inside,~
9890   columns-type,~
9891   columns-width,~
9892   corners,~
9893   create-extra-nodes,~
9894   create-medium-nodes,~
9895   create-large-nodes,~
9896   extra-left-margin,~
9897   extra-right-margin,~
9898   first-col,~
9899   first-row,~
9900   hlines,~
9901   hvlines,~
9902   hvlines-except-borders,~
9903   l,~
9904   last-col,~
9905   last-row,~
9906   left-margin,~
9907   light-syntax,~
9908   name,~
9909   nullify-dots,~
9910   pgf-node-code,~
9911   r,~
9912   renew-dots,~
9913   respect-arraystretch,~
9914   right-margin,~
9915   rounded-corners,~
9916   rules~(with~the~subkeys~'color'~and~'width'),~
9917   small,~
9918   t,~
9919   vlines,~
9920   xdots/color,~
9921   xdots/shorten-start,~
9922   xdots/shorten-end,~
9923   xdots/shorten-and~
9924   xdots/line-style.
9925 }
9926 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9927 {
9928   Unknown~key.\\
9929   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9930   \{NiceTabular\}. \\
9931   That~key~will~be~ignored. \\
9932   \c_@@_available_keys_str
9933 }
9934 {
9935   The~available~keys~are~(in~alphabetic~order):~
9936   b,~
9937   baseline,~
9938   c,~

```

```

9939 caption,~
9940 cell-space-bottom-limit,~
9941 cell-space-limits,~
9942 cell-space-top-limit,~
9943 code-after,~
9944 code-for-first-col,~
9945 code-for-first-row,~
9946 code-for-last-col,~
9947 code-for-last-row,~
9948 color-inside,~
9949 columns-width,~
9950 corners,~
9951 custom-line,~
9952 create-extra-nodes,~
9953 create-medium-nodes,~
9954 create-large-nodes,~
9955 extra-left-margin,~
9956 extra-right-margin,~
9957 first-col,~
9958 first-row,~
9959 hlines,~
9960 hvlines,~
9961 hvlines-except-borders,~
9962 label,~
9963 last-col,~
9964 last-row,~
9965 left-margin,~
9966 light-syntax,~
9967 name,~
9968 notes~(several~subkeys),~
9969 nullify-dots,~
9970 pgf-node-code,~
9971 renew-dots,~
9972 respect-arraystretch,~
9973 right-margin,~
9974 rounded-corners,~
9975 rules~(with~the~subkeys~'color'~and~'width'),~
9976 short-caption,~
9977 t,~
9978 tabularnote,~
9979 vlines,~
9980 xdots/color,~
9981 xdots/shorten-start,~
9982 xdots/shorten-end,~
9983 xdots/shorten-and~
9984 xdots/line-style.
9985 }

9986 \@@_msg_new:nnn { Duplicate~name }
9987 {
9988   Duplicate~name.\\
9989   The~name~'\l_keys_value_tl'~is~already-used-and-you-shouldn't-use~
9990   the~same~environment~name~twice.~You~can~go~on,~but,~
9991   maybe,~you~will~have~incorrect~results~especially~
9992   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9993   message~again,~use~the~key~'allow-duplicate-names'~in~
9994   '\token_to_str:N \NiceMatrixOptions'.\\
9995   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9996     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9997 }
9998 {
9999   The~names~already~defined~in~this~document~are:~
10000   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10001 }

```

```

10002 \@@_msg_new:nn { Option~auto~for~columns~width }
10003 {
10004   Erroneous~use.\\
10005   You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
10006   That~key~will~be~ignored.
10007 }
10008 \@@_msg_new:nn { NiceTabularX~without~X }
10009 {
10010   NiceTabularX~without~X.\\
10011   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10012   However,~you~can~go~on.
10013 }
10014 \@@_msg_new:nn { Preamble~forgotten }
10015 {
10016   Preamble~forgotten.\\
10017   You~have~probably~forgotten~the~preamble~of~your~
10018   \@@_full_name_env:. \\
10019   This~error~is~fatal.
10020 }

```

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>	72
14	The environment <code>{NiceMatrix}</code> and its variants	90
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	91
16	After the construction of the array	92
17	We draw the dotted lines	98
18	The actual instructions for drawing the dotted lines with Tikz	112
19	User commands available in the new environments	117
20	The command <code>\line</code> accessible in code-after	123
21	The command <code>\RowStyle</code>	125
22	Colors of cells, rows and columns	128
23	The vertical and horizontal rules	139
24	The empty corners	154
25	The environment <code>{NiceMatrixBlock}</code>	156
26	The extra nodes	158
27	The blocks	162
28	How to draw the dotted lines transparently	182
29	Automatic arrays	182
30	The redefinition of the command <code>\dotfill</code>	183

31	The command <code>\diagbox</code>	184
32	The keyword <code>\CodeAfter</code>	185
33	The delimiters in the preamble	186
34	The command <code>\SubMatrix</code>	187
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	195
36	The command <code>TikzEveryCell</code>	198
37	The command <code>\ShowCellNames</code>	200
38	We process the options at package loading	203
39	About the package underscore	204
40	Error messages of the package	205