

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

November 3, 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 6.25a of `nicematrix`, at the date of 2023/11/03.

```

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_pgful@check@rerun \pgfutil@check@rerun

```

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

```

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_pgful@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_pgfpntanchor \pgfpntanchor

```

```

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@mc@cell\relax
302 \@temptokena{#1} % \@tempswatrue
303 % \@whiles\if@tempswa\fi{\@tempswafalse\the\NC@list}%
304 \count@m@one
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 \bool_if:NTF \c_@@_revtex_bool
327 { \cs_new_protected:Npn \@_redefine_mkpream: { } }
328 {
329 \cs_new_protected:Npn \@_redefine_mkpream:
330 { \cs_set_eq:NN \@mkpream \@_mkpream: }
331 }

```

```

332 \cs_new_protected:Npn \@_mkpream_colortbl: { }
333 \hook_gput_code:nnn { begindocument } { . }
334 {
335 \IfPackageLoadedTF { colortbl }
336 {
337 \cs_set_protected:Npn \@_mkpream_colortbl:
338 {

```

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.

```

339         \cs_set_eq:NN \CT@setup \relax
340         \cs_set_eq:NN \CT@color \relax
341         \cs_set_eq:NN \CT@do@color \relax
342         \cs_set_eq:NN \color \relax
343         \cs_set_eq:NN \CT@column@color \relax
344         \cs_set_eq:NN \CT@row@color \relax
345         \cs_set_eq:NN \CT@cell@color \relax
346     }
347 }
348 { }
349 }
```

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.

```

350 \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).

```

351 \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.

```

352 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
353 { \int_use:N \g_@@_env_int }
```

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

```

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

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

```

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

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

```

357 \bool_new:N \g_@@_delims_bool
358 \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.

```

359 \bool_new:N \l_@@_preamble_bool
360 \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.

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

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

```
362 \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.

```
363 \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).

```
364 \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.).

```
365 \dim_new:N \l_@@_col_width_dim
```

```
366 \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.

```
367 \int_new:N \g_@@_row_total_int
```

```
368 \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).

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

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

```
370 \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.

```
371 \str_new:N \l_@@_hpos_cell_str
```

```
372 \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`.

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

Idem for the mono-row blocks.

```
374 \dim_new:N \g_@@_blocks_ht_dim
```

```
375 \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`.

```
376 \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}`).

```
377 \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`.

```
378 \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.

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

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

```
380 \bool_new:N \l_@@_notes_detect_duplicates_bool
381 \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.

```
382 \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).

```
383 \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).

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

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

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

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

```
386 \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.

```
387 \bool_new:N \l_@@_X_bool
388 \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 }`).

```
389 \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.

```
390 \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.

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

```

392 \tl_new:N \g_@@_left_delim_tl
393 \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}`).

```

394 \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`).

```

395 \tl_new:N \g_@@_array_preamble_tl

```

For `\multicolumn`.

```

396 \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`.

```

397 \tl_new:N \l_@@_columns_type_tl
398 \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 `..`.

```

399 \tl_new:N \l_@@_xdots_down_tl
400 \tl_new:N \l_@@_xdots_up_tl
401 \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`).

```

402 \seq_new:N \g_@@_rowlistcolors_seq

```

```

403 \cs_new_protected:Npn \@@_test_if_math_mode:
404 {
405   \if_mode_math: \else:
406     \@@_fatal:n { Outside~math~mode }
407   \fi:
408 }

```

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.

```

409 \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”.

```

410 \colorlet { nicematrix-last-col } { . }
411 \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`).

```

412 \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*.

```

413 \tl_new:N \g_@@_com_or_env_str
414 \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).

```

415 \cs_new:Npn \@@_full_name_env:
416 {
417   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
418     { command \space \c_backslash_str \g_@@_name_env_str }
419     { environment \space \{ \g_@@_name_env_str \} }
420 }
```

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

```

421 \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.

```

422 \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.

```

423
```

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

```

424 \tl_new:N \g_@@_pre_code_before_tl
425 \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.

```

426 \tl_new:N \g_@@_pre_code_after_tl
427 \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.

```

428 \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.

```

429 \int_new:N \l_@@_old_iRow_int
430 \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).

```

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

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

```

432 \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.

```
433 \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.

```
434 \bool_new:N \l_@@_X_columns_aux_bool
435 \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.

```
436 \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).

```
437 \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).

```
438 \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`).

```
439 \tl_new:N \l_@@_code_before_tl
440 \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).

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

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

```
442 \dim_new:N \l_@@_x_initial_dim
443 \dim_new:N \l_@@_y_initial_dim
444 \dim_new:N \l_@@_x_final_dim
445 \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.

```
446 \dim_zero_new:N \l_@@_tmpc_dim
447 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
448 \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”.

```
449 \dim_new:N \g_@@_width_last_col_dim
450 \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.

```
451 \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!).

```
452 \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}`.

```
453 \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.

```
454 \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.

```
455 \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`).

```
456 \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.

```
457 \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).

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

```
459 \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).

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

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

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

```
463 \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.

```
464 \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).

```
465 \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`.

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

```
472 \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`.

```
473 \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.

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

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

```
475 \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`).

```
476 \str_new:N \l_@@_hpos_block_str
477 \str_set:Nn \l_@@_hpos_block_str { c }
478 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

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

```
479 \str_new:N \l_@@_vpos_of_block_str
480 \str_set:Nn \l_@@_vpos_of_block_str { c }
```

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

```
481 \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).

```
482 \bool_new:N \l_@@_vlines_block_bool
483 \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.

```

484 \int_new:N \g_@@_block_box_int

485 \dim_new:N \l_@@_submatrix_extra_height_dim
486 \dim_new:N \l_@@_submatrix_left_xshift_dim
487 \dim_new:N \l_@@_submatrix_right_xshift_dim
488 \clist_new:N \l_@@_hlines_clist
489 \clist_new:N \l_@@_vlines_clist
490 \clist_new:N \l_@@_submatrix_hlines_clist
491 \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}`).

```

492 \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.

```

493 \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`).

```

494 \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.

```

495 \int_new:N \l_@@_first_row_int
496 \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.

```

497 \int_new:N \l_@@_first_col_int
498 \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).

```

499 \int_new:N \l_@@_last_row_int
500 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
502 \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`.

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

```
505 \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`).

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

Some utilities

```
507 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
508 {
509   \tl_set:Nn \l_tmpa_tl { #1 }
510   \tl_set:Nn \l_tmpb_tl { #2 }
511 }
```

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.

```
512 \cs_new_protected:Npn \@@_expand_clist:N #1
513 {
514   \clist_if_in:NnF #1 { all }
515   {
516     \clist_clear:N \l_tmpa_clist
517     \clist_map_inline:Nn #1
```

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.

```

518     {
519         \tl_if_in:nnTF { ##1 } { - }
520         { \@@_cut_on_hyphen:w ##1 \q_stop }
521         {
522             \tl_set:Nn \l_tmpa_tl { ##1 }
523             \tl_set:Nn \l_tmpb_tl { ##1 }
524         }
525         \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
526         { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
527     }
528     \tl_set_eq:NN #1 \l_tmpa_clist
529 }
530 }

```

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.

```

531 \hook_gput_code:nnn { begindocument } { . }
532 {
533     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
534     \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
535     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
536 }

```

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 tabularnote}`. The first composante is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker `\c_novalue_tl`).

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

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

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

```
537 \newcounter { tabularnote }
538 \seq_new:N \g_@@_notes_seq
539 \seq_new:N \g_@@_notes_in_caption_seq
```

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

```
540 \tl_new:N \g_@@_tabularnote_tl
```

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

```
541 \seq_new:N \l_@@_notes_labels_seq
542 \newcounter{nicematrix_draft}
543 \cs_new_protected:Npn \@@_notes_format:n #1
544 {
545   \setcounter { nicematrix_draft } { #1 }
546   \@@_notes_style:n { nicematrix_draft }
547 }
```

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

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

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

```
549 \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`.

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

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

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

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

```
552 \hook_gput_code:nnn { begindocument } { . }
553 {
554   \IfPackageLoadedTF { enumitem }
555   {
```

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.

```

556     \newlist { tabularnotes } { enumerate } { 1 }
557     \setlist [ tabularnotes ]
558     {
559         topsep = Opt ,
560         noitemsep ,
561         leftmargin = * ,
562         align = left ,
563         labelsep = Opt ,
564         label =
565             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
566     }
567     \newlist { tabularnotes* } { enumerate* } { 1 }
568     \setlist [ tabularnotes* ]
569     {
570         afterlabel = \nobreak ,
571         itemjoin = \quad ,
572         label =
573             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
574     }

```

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`.

```

575     \NewDocumentCommand \tabularnote { o m }
576     {
577         \bool_if:nT { \cs_if_exist_p:N \@capttype || \l_@@_in_env_bool }
578         {
579             \bool_if:nTF { ! \l_@@_tabular_bool && \l_@@_in_env_bool }
580             { \@@_error:n { tabularnote~forbidden } }
581             {
582                 \bool_if:NTF \l_@@_in_caption_bool
583                 { \@@_tabularnote_caption:nn { #1 } { #2 } }
584                 { \@@_tabularnote:nn { #1 } { #2 } }
585             }
586         }
587     }
588 }
589 {
590     \NewDocumentCommand \tabularnote { o m }
591     {
592         \@@_error_or_warning:n { enumitem~not~loaded }
593         \@@_gredirect_none:n { enumitem~not~loaded }
594     }
595 }
596 }

```

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`.

```

597 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
598 {

```

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).

```

599     \int_zero:N \l_tmpa_int
600     \bool_if:NT \l_@@_notes_detect_duplicates_bool

```

601 {

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`.

```

602   \seq_map_indexed_inline:Nn \g_@@_notes_seq
603   {
604     \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
605     {
606       \int_set:Nn \l_tmpa_int { ##1 }
607       \seq_map_break:
608     }
609   }
610   \int_if_zero:nF \l_tmpa_int
611   { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
612 }
613 \int_if_zero:nT \l_tmpa_int
614 {
615   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
616   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
617 }
618 \seq_put_right:Nx \l_@@_notes_labels_seq
619 {
620   \tl_if_novalue:nTF { #1 }
621   {
622     \@@_notes_format:n
623     {
624       \int_eval:n
625       {
626         \int_if_zero:nTF \l_tmpa_int
627         \c@tabularnote
628         \l_tmpa_int
629       }
630     }
631   }
632   { #1 }
633 }
634 \peek_meaning:NF \tabularnote
635 {

```

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`.

```

636   \hbox_set:Nn \l_tmpa_box
637   {

```

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

```

638   \@@_notes_label_in_tabular:n
639   {
640     \seq_use:Nnnn
641     \l_@@_notes_labels_seq { , } { , } { , }
642   }
643 }

```

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

```

644   \int_gsub:Nn \c@tabularnote { 1 }
645   \int_set_eq:NN \l_tmpa_int \c@tabularnote
646   \refstepcounter { tabularnote }
647   \int_compare:nNnT \l_tmpa_int = \c@tabularnote

```

```

648     { \int_gincr:N \c@tabularnote }
649     \seq_clear:N \l_@@_notes_labels_seq
650     \bool_lazy_or:nnTF
651     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { c } }
652     { \str_if_eq_p:Vn \l_@@_hpos_cell_str { r } }
653     {
654         \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).

```

655         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
656     }
657     { \box_use:N \l_tmpa_box }
658 }
659 }

```

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`.

```

660 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
661 {
662     \bool_if:NTF \g_@@_caption_finished_bool
663     {
664         \int_compare:nNnT
665         \c@tabularnote = \g_@@_notes_caption_int
666         { \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`!

```

667     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
668     { \@@_error:n { Identical~notes~in~caption } }
669 }
670 {

```

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

```

671     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
672     {

```

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.

```

673         \bool_gset_true:N \g_@@_caption_finished_bool
674         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
675         \int_gzero:N \c@tabularnote
676     }
677     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
678 }

```

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!

```

679     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
680     \seq_put_right:Nx \l_@@_notes_labels_seq
681     {
682         \tl_if_novalue:NTF { #1 }
683         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
684         { #1 }
685     }
686     \peek_meaning:NF \tabularnote
687     {

```



```

688 \@@_notes_label_in_tabular:n
689 { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
690 \seq_clear:N \l_@@_notes_labels_seq
691 }
692 }
693 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
694 { \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.

```

695 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
696 {
697   \begin { pgfscope }
698   \pgfset
699   {
700     inner~sep = \c_zero_dim ,
701     minimum~size = \c_zero_dim
702   }
703   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
704   \pgfnode
705   { rectangle }
706   { center }
707   {
708     \vbox_to_ht:nn
709     { \dim_abs:n { #5 - #3 } }
710     {
711       \vfill
712       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
713     }
714   }
715   { #1 }
716   { }
717   \end { pgfscope }
718 }

```

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.

```

719 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
720 {
721   \begin { pgfscope }
722   \pgfset
723   {
724     inner~sep = \c_zero_dim ,
725     minimum~size = \c_zero_dim
726   }
727   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
728   \pgfpointdiff { #3 } { #2 }
729   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
730   \pgfnode
731   { rectangle }
732   { center }
733   {
734     \vbox_to_ht:nn
735     { \dim_abs:n \l_tmpb_dim }
736     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }

```

```

737     }
738     { #1 }
739     { }
740     \end { pgfscope }
741 }

```

8 The options

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

```

742 \tl_new:N \l_@@_caption_tl
743 \tl_new:N \l_@@_short_caption_tl
744 \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).

```

745 \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.

```

746 \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`.

```

747 \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`).

```

748 \dim_new:N \l_@@_cell_space_top_limit_dim
749 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

750 \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.

```

751 \dim_new:N \l_@@_xdots_inter_dim
752 \hook_gput_code:nnn { begindocument } { . }
753 { \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).

```

754 \dim_new:N \l_@@_xdots_shorten_start_dim
755 \dim_new:N \l_@@_xdots_shorten_end_dim
756 \hook_gput_code:nnn { begindocument } { . }
757 {
758   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
759   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
760 }

```

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.

```
761 \dim_new:N \l_@@_xdots_radius_dim
762 \hook_gput_code:nnn { begindocument } { . }
763 { \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.

```
764 \tl_new:N \l_@@_xdots_line_style_tl
765 \tl_const:Nn \c_@@_standard_tl { standard }
766 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
767 \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).

```
768 \tl_new:N \l_@@_baseline_tl
769 \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`).

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

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

```
771 \bool_new:N \l_@@_parallelize_diags_bool
772 \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.

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

```
774 \dim_new:N \l_@@_notes_above_space_dim
775 \hook_gput_code:nnn { begindocument } { . }
776 { \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.).

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

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

```
778 \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).

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

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

```
780 \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.

```
781 \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”.

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

```
783 \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).

```
784 \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`).

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

```
786 \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`.

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

```
788 \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.

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

```
790 \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 “:”.

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

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

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

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

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

```

794 \keys_define:nn { NiceMatrix / xdots }
795 {
796   shorten-start .code:n =
797     \hook_gput_code:nnn { begindocument } { . }
798     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
799   shorten-end .code:n =
800     \hook_gput_code:nnn { begindocument } { . }
801     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
802   shorten-start .value_required:n = true ,
803   shorten-end .value_required:n = true ,
804   shorten .code:n =
805     \hook_gput_code:nnn { begindocument } { . }
806     {
807       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
808       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
809     } ,
810   shorten .value_required:n = true ,
811   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
812   horizontal-labels .default:n = true ,
813   line-style .code:n =
814     {
815       \bool_lazy_or:nnTF
816         { \cs_if_exist_p:N \tikzpicture }
817         { \str_if_eq_p:nn { #1 } { standard } }
818         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
819         { \@@_error:n { bad~option~for~line~style } }
820     } ,
821   line-style .value_required:n = true ,
822   color .tl_set:N = \l_@@_xdots_color_tl ,
823   color .value_required:n = true ,
824   radius .code:n =
825     \hook_gput_code:nnn { begindocument } { . }
826     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
827   radius .value_required:n = true ,
828   inter .code:n =
829     \hook_gput_code:nnn { begindocument } { . }
830     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
831   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 \wedge , $_$ and \therefore . We use \tl_put_right:Nn and not \tl_set:Nn (or \tl_set:N) because we don't want a direct use of $\text{up}=\dots$ erased by a absent $\wedge\{\dots\}$.

```

832   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
833   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
834   middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

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

```

835   draw-first .code:n = \prg_do_nothing: ,
836   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
837 }

```

```

838 \keys_define:nn { NiceMatrix / rules }
839 {
840   color .tl_set:N = \l_@@_rules_color_tl ,
841   color .value_required:n = true ,
842   width .dim_set:N = \arrayrulewidth ,
843   width .value_required:n = true ,
844   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
845 }

```

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

```

846 \keys_define:nn { NiceMatrix / Global }
847 {
848   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
849   rounded-corners .default:n = 4 pt ,
850   custom-line .code:n = \@@_custom_line:n { #1 } ,
851   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
852   rules .value_required:n = true ,
853   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
854   standard-cline .default:n = true ,
855   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
856   cell-space-top-limit .value_required:n = true ,
857   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
858   cell-space-bottom-limit .value_required:n = true ,
859   cell-space-limits .meta:n =
860   {
861     cell-space-top-limit = #1 ,
862     cell-space-bottom-limit = #1 ,
863   } ,
864   cell-space-limits .value_required:n = true ,
865   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
866   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
867   light-syntax .default:n = true ,
868   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
869   end-of-row .value_required:n = true ,
870   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
871   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
872   last-row .int_set:N = \l_@@_last_row_int ,
873   last-row .default:n = -1 ,
874   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
875   code-for-first-col .value_required:n = true ,
876   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
877   code-for-last-col .value_required:n = true ,
878   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
879   code-for-first-row .value_required:n = true ,
880   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
881   code-for-last-row .value_required:n = true ,
882   hlines .clist_set:N = \l_@@_hlines_clist ,
883   vlines .clist_set:N = \l_@@_vlines_clist ,
884   hlines .default:n = all ,
885   vlines .default:n = all ,
886   vlines-in-sub-matrix .code:n =
887   {
888     \tl_if_single_token:nTF { #1 }
889     {
890       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
891       { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

892     { \cs_set_eq:cN { @@_#1 } \@@_make_preamble_vlism:n }
893   }
894   { \@@_error:n { One~letter~allowed } }
895 } ,
896 vlines-in-sub-matrix .value_required:n = true ,
897 hvlines .code:n =
898 {
899   \bool_set_true:N \l_@@_hvlines_bool
900   \clist_set:Nn \l_@@_vlines_clist { all }
901   \clist_set:Nn \l_@@_hlines_clist { all }
902 } ,
903 hvlines-except-borders .code:n =
904 {

```

```

905     \clist_set:Nn \l_@@_vlines_clist { all }
906     \clist_set:Nn \l_@@_hlines_clist { all }
907     \bool_set_true:N \l_@@_hvlines_bool
908     \bool_set_true:N \l_@@_except_borders_bool
909   } ,
910   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.

```

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

```

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

```

934 \keys_define:nn { NiceMatrix / Env }
935 {
936   corners .clist_set:N = \l_@@_corners_clist ,
937   corners .default:n = { NW , SW , NE , SE } ,
938   code-before .code:n =
939     {
940       \tl_if_empty:nF { #1 }
941       {
942         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
943         \bool_set_true:N \l_@@_code_before_bool
944       }
945     } ,
946   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}`.

```

947   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
948   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
949   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
950   baseline .tl_set:N = \l_@@_baseline_tl ,
951   baseline .value_required:n = true ,
952   columns-width .code:n =
953     \tl_if_eq:nnTF { #1 } { auto }
954     { \bool_set_true:N \l_@@_auto_columns_width_bool }
955     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
956   columns-width .value_required:n = true ,
957   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.

```

958 \legacy_if:nF { measuring@ }
959 {
960   \str_set:Nx \l_tmpa_str { #1 }
961   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
962   { \@@_error:nn { Duplicate~name } { #1 } }
963   { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
964   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
965 } ,
966 name .value_required:n = true ,
967 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
968 code-after .value_required:n = true ,
969 color-inside .code:n =
970   \bool_set_true:N \l_@@_color_inside_bool
971   \bool_set_true:N \l_@@_code_before_bool ,
972 color-inside .value_forbidden:n = true ,
973 colortbl-like .meta:n = color-inside
974 }
975 \keys_define:nn { NiceMatrix / notes }
976 {
977   para .bool_set:N = \l_@@_notes_para_bool ,
978   para .default:n = true ,
979   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
980   code-before .value_required:n = true ,
981   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
982   code-after .value_required:n = true ,
983   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
984   bottomrule .default:n = true ,
985   style .cs_set:Np = \@@_notes_style:n #1 ,
986   style .value_required:n = true ,
987   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
988   label-in-tabular .value_required:n = true ,
989   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
990   label-in-list .value_required:n = true ,
991   enumitem-keys .code:n =
992   {
993     \hook_gput_code:nnn { begindocument } { . }
994     {
995       \IfPackageLoadedTF { enumitem }
996       { \setlist* [ tabularnotes ] { #1 } }
997       { }
998     }
999   } ,
1000   enumitem-keys .value_required:n = true ,
1001   enumitem-keys-para .code:n =
1002   {
1003     \hook_gput_code:nnn { begindocument } { . }
1004     {
1005       \IfPackageLoadedTF { enumitem }
1006       { \setlist* [ tabularnotes* ] { #1 } }
1007       { }
1008     }
1009   } ,
1010   enumitem-keys-para .value_required:n = true ,
1011   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1012   detect-duplicates .default:n = true ,
1013   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1014 }
1015 \keys_define:nn { NiceMatrix / delimiters }
1016 {
1017   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1018   max-width .default:n = true ,

```



```

1019   color .tl_set:N = \l_@@_delimiters_color_tl ,
1020   color .value_required:n = true ,
1021 }

```

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

```

1022 \keys_define:nn { NiceMatrix }
1023 {
1024   NiceMatrixOptions .inherit:n =
1025     { NiceMatrix / Global } ,
1026   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1027   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1028   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1029   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1030   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1031   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1032   CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1033   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1034   NiceMatrix .inherit:n =
1035     {
1036       NiceMatrix / Global ,
1037       NiceMatrix / Env ,
1038     } ,
1039   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1040   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1041   NiceTabular .inherit:n =
1042     {
1043       NiceMatrix / Global ,
1044       NiceMatrix / Env
1045     } ,
1046   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1047   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1048   NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1049   NiceArray .inherit:n =
1050     {
1051       NiceMatrix / Global ,
1052       NiceMatrix / Env ,
1053     } ,
1054   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1055   NiceArray / rules .inherit:n = NiceMatrix / rules ,
1056   pNiceArray .inherit:n =
1057     {
1058       NiceMatrix / Global ,
1059       NiceMatrix / Env ,
1060     } ,
1061   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1062   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1063 }

```

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

```

1064 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1065 {
1066   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1067   delimiters / color .value_required:n = true ,
1068   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1069   delimiters / max-width .default:n = true ,
1070   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1071   delimiters .value_required:n = true ,
1072   width .dim_set:N = \l_@@_width_dim ,
1073   width .value_required:n = true ,
1074   last-col .code:n =

```

```

1075 \tl_if_empty:nF { #1 }
1076 { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1077 \int_zero:N \l_@@_last_col_int ,
1078 small .bool_set:N = \l_@@_small_bool ,
1079 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.

```

1080 renew-matrix .code:n = \@@_renew_matrix: ,
1081 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}`.

```

1082 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.

```

1083 columns-width .code:n =
1084 \tl_if_eq:nnTF { #1 } { auto }
1085 { \@@_error:n { Option~auto~for~columns-width } }
1086 { \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.

```

1087 allow-duplicate-names .code:n =
1088 \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1089 allow-duplicate-names .value_forbidden:n = true ,
1090 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1091 notes .value_required:n = true ,
1092 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1093 sub-matrix .value_required:n = true ,
1094 matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1095 matrix / columns-type .value_required:n = true ,
1096 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1097 caption-above .default:n = true ,
1098 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1099 }

```

`\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.

```

1100 \NewDocumentCommand \NiceMatrixOptions { m }
1101 { \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.

```

1102 \keys_define:nn { NiceMatrix / NiceMatrix }
1103 {
1104   last-col .code:n = \tl_if_empty:nTF { #1 }
1105   {
1106     \bool_set_true:N \l_@@_last_col_without_value_bool
1107     \int_set:Nn \l_@@_last_col_int { -1 }
1108   }
1109   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1110   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1111   columns-type .value_required:n = true ,
1112   l .meta:n = { columns-type = l } ,
1113   r .meta:n = { columns-type = r } ,
1114   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1115   delimiters / color .value_required:n = true ,

```

```

1116 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1117 delimiters / max-width .default:n = true ,
1118 delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1119 delimiters .value_required:n = true ,
1120 small .bool_set:N = \l_@@_small_bool ,
1121 small .value_forbidden:n = true ,
1122 unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1123 }

```

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

```

1124 \keys_define:nn { NiceMatrix / NiceArray }
1125 {

```

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.

```

1126 small .bool_set:N = \l_@@_small_bool ,
1127 small .value_forbidden:n = true ,
1128 last-col .code:n = \tl_if_empty:nF { #1 }
1129 { \@@_error:n { last-col~non~empty~for~NiceArray } }
1130 \int_zero:N \l_@@_last_col_int ,
1131 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1132 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1133 unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1134 }

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

```

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

```

1154 \keys_define:nn { NiceMatrix / NiceTabular }
1155 {

```

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.

```

1156 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1157 \bool_set_true:N \l_@@_width_used_bool ,
1158 width .value_required:n = true ,
1159 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1160 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1161 tabularnote .value_required:n = true ,
1162 caption .tl_set:N = \l_@@_caption_tl ,
1163 caption .value_required:n = true ,
1164 short-caption .tl_set:N = \l_@@_short_caption_tl ,

```

```

1165 short-caption .value_required:n = true ,
1166 label .tl_set:N = \l_@@_label_tl ,
1167 label .value_required:n = true ,
1168 last-col .code:n = \tl_if_empty:nF {#1}
1169 { \@@_error:n { last-col~non-empty~for~NiceArray } }
1170 \int_zero:N \l_@@_last_col_int ,
1171 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1172 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1173 unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1174 }

```

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
```

```

1175 \keys_define:nn { NiceMatrix / CodeAfter }
1176 {
1177   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1178   delimiters / color .value_required:n = true ,
1179   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1180   rules .value_required:n = true ,
1181   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1182   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1183   sub-matrix .value_required:n = true ,
1184   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1185 }

```

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}`).

```

1186 \cs_new_protected:Npn \@@_cell_begin:w
1187 {

```

`\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.

```
1188   \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).

```
1189   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1190   \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.

```

1191   \int_compare:nNnT \c@jCol = 1
1192     { \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).

```

1193   \hbox_set:Nw \l_@@_cell_box
1194   \bool_if:NF \l_@@_tabular_bool
1195   {
1196     \c_math_toggle_token
1197     \bool_if:NT \l_@@_small_bool \scriptstyle
1198   }
1199   \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.

```

1200   \int_if_zero:nTF \c@iRow
1201   {
1202     \int_compare:nNnT \c@jCol > 0
1203     {
1204       \l_@@_code_for_first_row_tl
1205       \xglobal \colorlet { nicematrix-first-row } { . }
1206     }
1207   }
1208   {
1209     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1210     {
1211       \l_@@_code_for_last_row_tl
1212       \xglobal \colorlet { nicematrix-last-row } { . }
1213     }
1214   }
1215 }

```

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.

```

1216 \cs_new_protected:Npn \@@_begin_of_row:
1217 {
1218   \int_gincr:N \c@iRow
1219   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1220   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1221   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1222   \pgfpicture
1223   \pgfrememberpicturepositiononpagetrue
1224   \pgfcoordinate
1225   { \@@_env: - row - \int_use:N \c@iRow - base }
1226   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1227   \str_if_empty:NF \l_@@_name_str
1228   {
1229     \pgfnodealias
1230     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1231     { \@@_env: - row - \int_use:N \c@iRow - base }
1232   }
1233   \endpgfpicture
1234 }

```

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.

```

1235 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1236 {
1237   \int_if_zero:nTF \c@iRow

```

```

1238 {
1239   \dim_gset:Nn \g_@@_dp_row_zero_dim
1240   { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1241   \dim_gset:Nn \g_@@_ht_row_zero_dim
1242   { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1243 }
1244 {
1245   \int_compare:nNnT \c@iRow = 1
1246   {
1247     \dim_gset:Nn \g_@@_ht_row_one_dim
1248     { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1249   }
1250 }
1251 }
1252 \cs_new_protected:Npn \@@_rotate_cell_box:
1253 {
1254   \box_rotate:Nn \l_@@_cell_box { 90 }
1255   \bool_if:NTF \g_@@_rotate_c_bool
1256   {
1257     \hbox_set:Nn \l_@@_cell_box
1258     {
1259       \c_math_toggle_token
1260       \vcenter { \box_use:N \l_@@_cell_box }
1261       \c_math_toggle_token
1262     }
1263   }
1264   {
1265     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1266     {
1267       \vbox_set_top:Nn \l_@@_cell_box
1268       {
1269         \vbox_to_zero:n { }
1270         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1271         \box_use:N \l_@@_cell_box
1272       }
1273     }
1274   }
1275   \bool_gset_false:N \g_@@_rotate_bool
1276   \bool_gset_false:N \g_@@_rotate_c_bool
1277 }
1278 \cs_new_protected:Npn \@@_adjust_size_box:
1279 {
1280   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1281   {
1282     \box_set_wd:Nn \l_@@_cell_box
1283     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1284     \dim_gzero:N \g_@@_blocks_wd_dim
1285   }
1286   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1287   {
1288     \box_set_dp:Nn \l_@@_cell_box
1289     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1290     \dim_gzero:N \g_@@_blocks_dp_dim
1291   }
1292   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1293   {
1294     \box_set_ht:Nn \l_@@_cell_box
1295     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1296     \dim_gzero:N \g_@@_blocks_ht_dim
1297   }
1298 }
1299 \cs_new_protected:Npn \@@_cell_end:
1300 {

```

```

1301 \@@_math_toggle_token:
1302 \hbox_set_end:
1303 \@@_cell_end_i:
1304 }
1305 \cs_new_protected:Npn \@@_cell_end_i:
1306 {

```

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.

```

1307 \g_@@_cell_after_hook_tl
1308 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1309 \@@_adjust_size_box:
1310 \box_set_ht:Nn \l_@@_cell_box
1311 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1312 \box_set_dp:Nn \l_@@_cell_box
1313 { \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”).

```

1314 \@@_update_max_cell_width:

```

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

```

1315 \@@_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.

```

1316 \bool_if:NTF \g_@@_empty_cell_bool
1317 { \box_use_drop:N \l_@@_cell_box }
1318 {
1319 \bool_lazy_or:nnTF
1320 \g_@@_not_empty_cell_bool
1321 { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1322 \@@_node_for_cell:
1323 { \box_use_drop:N \l_@@_cell_box }
1324 }
1325 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1326 \bool_gset_false:N \g_@@_empty_cell_bool
1327 \bool_gset_false:N \g_@@_not_empty_cell_bool
1328 }

```

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

```

1329 \cs_new_protected:Npn \@@_update_max_cell_width:
1330 {
1331   \dim_gset:Nn \g_@@_max_cell_width_dim
1332   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1333 }

```

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`).

```

1334 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1335 {
1336   \@@_math_toggle_token:
1337   \hbox_set_end:
1338   \bool_if:NF \g_@@_rotate_bool
1339   {
1340     \hbox_set:Nn \l_@@_cell_box
1341     {
1342       \makebox [ \l_@@_col_width_dim ] [ s ]
1343       { \hbox_unpack_drop:N \l_@@_cell_box }
1344     }
1345   }
1346   \@@_cell_end_i:
1347 }

```

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

```

1348 \pgfset
1349 {
1350   nicematrix / cell-node /.style =
1351   {
1352     inner~sep = \c_zero_dim ,
1353     minimum~width = \c_zero_dim
1354   }
1355 }
1356 \cs_new_protected:Npn \@@_node_for_cell:
1357 {
1358   \pgfpicture
1359   \pgfsetbaseline \c_zero_dim
1360   \pgfrememberpicturepositiononpagetrue
1361   \pgfset { nicematrix / cell-node }
1362   \pgfnode
1363   { rectangle }
1364   { base }
1365   {

```

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).

```

1366   \set@color
1367   \box_use_drop:N \l_@@_cell_box
1368 }
1369 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1370 { \l_@@_pgf_node_code_tl }
1371 \str_if_empty:NF \l_@@_name_str
1372 {
1373   \pgfnodealias
1374   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1375   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1376 }
1377 \endpgfpicture
1378 }

```


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.

```

1379 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1380 {
1381   \cs_new_protected:Npn \@@_patch_node_for_cell:
1382   {
1383     \hbox_set:Nn \l_@@_cell_box
1384     {
1385       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1386       \hbox_overlap_left:n
1387       {
1388         \pgfsys@markposition
1389         { \@@_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.

```

1390       #1
1391     }
1392     \box_use:N \l_@@_cell_box
1393     \box_move_down:nn { \box_dp: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 - SE }
1398     }
1399     #1
1400   }
1401 }
1402 }

```

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

```

1403 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1404 {
1405   \@@_patch_node_for_cell:n
1406   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1407 }
1408 { \@@_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).

```

1409 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1410 {
1411   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx

```

```

1412 { g_@@_ #2 _ lines _ t1 }
1413 {
1414   \use:c { @@ _ draw _ #2 : nnn }
1415   { \int_use:N \c@iRow }
1416   { \int_use:N \c@jCol }
1417   { \exp_not:n { #3 } }
1418 }
1419 }

```

```

1420 \cs_new_protected:Npn \@@_array:
1421 {

```

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).

```

1422 \@@_redefine_mkpream:
1423 \dim_set:Nn \col@sep
1424 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1425 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1426 { \cs_set_nopar:Npn \@halignto { } }
1427 { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1428 \@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.

```

1429 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1430 }

```

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.

```

1431 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1432 \cs_new_protected:Npn \@@_create_row_node:
1433 {
1434   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1435   {
1436     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1437     \@@_create_row_node_i:
1438   }
1439 }

```

```

1440 \cs_new_protected:Npn \@@_create_row_node_i:
1441 {

```

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

```

1442 \hbox
1443 {
1444   \bool_if:NT \l_@@_code_before_bool
1445   {
1446     \vtop
1447     {
1448       \skip_vertical:N 0.5\arrayrulewidth
1449       \pgfsys@markposition
1450       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1451       \skip_vertical:N -0.5\arrayrulewidth
1452     }
1453   }
1454   \pgfpicture
1455   \pgfrememberpicturepositiononpagetrue
1456   \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }

```

```

1457         { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1458     \str_if_empty:NF \l_@@_name_str
1459     {
1460         \pgfnodealias
1461         { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1462         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1463     }
1464     \endpgfpicture
1465 }
1466 }

```

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

```

1467 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1468 \cs_new_protected:Npn \@@_everycr_i:
1469 {
1470     \int_gzero:N \c@jCol
1471     \bool_gset_false:N \g_@@_after_col_zero_bool
1472     \bool_if:NF \g_@@_row_of_col_done_bool
1473     {
1474         \@@_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).

```

1475     \tl_if_empty:NF \l_@@_hlines_clist
1476     {
1477         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1478         {
1479             \exp_args:NNe
1480             \clist_if_in:NnT
1481             \l_@@_hlines_clist
1482             { \int_eval:n { \c@iRow + 1 } }
1483         }
1484     }

```

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.

```

1485     \int_compare:nNnT \c@iRow > { -1 }
1486     {
1487         \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@`.

```

1488         { \hrule height \arrayrulewidth width \c_zero_dim }
1489     }
1490 }
1491 }
1492 }
1493 }

```

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

```

1494 \cs_set_protected:Npn \@@_renew_dots:
1495 {
1496     \cs_set_eq:NN \ldots \@@_Ldots
1497     \cs_set_eq:NN \cdots \@@_Cdots
1498     \cs_set_eq:NN \vdots \@@_Vdots
1499     \cs_set_eq:NN \ddots \@@_Ddots
1500     \cs_set_eq:NN \iddots \@@_Iddots
1501     \cs_set_eq:NN \dots \@@_Ldots
1502     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1503 }

```

```

1504 \cs_new_protected:Npn \@@_test_color_inside:
1505 {
1506   \bool_if:NF \l_@@_color_inside_bool
1507   {

```

We will issue an error only during the first run.

```

1508     \bool_if:NF \g_@@_aux_found_bool
1509     { \@@_error:n { without~color-inside } }
1510   }
1511 }

```

```

1512 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1513 \hook_gput_code:nnn { begindocument } { . }
1514 {
1515   \IfPackageLoadedTF { colortbl }
1516   {
1517     \cs_set_protected:Npn \@@_redefine_everycr:
1518     {
1519       \CT@everycr
1520       {
1521         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1522         \@@_everycr:
1523       }
1524     }
1525   }
1526   { }
1527 }

```

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.

```

1528 \hook_gput_code:nnn { begindocument } { . }
1529 {
1530   \IfPackageLoadedTF { booktabs }
1531   {
1532     \cs_new_protected:Npn \@@_patch_booktabs:
1533     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1534   }
1535   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1536 }

```

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

```

1537 \cs_new_protected:Npn \@@_pre_array_ii:
1538 {

```

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

```

1539   \int_gzero:N \g_@@_total_X_weight_int
1540   \@@_expand_clist:N \l_@@_hlines_clist
1541   \@@_expand_clist:N \l_@@_vlines_clist
1542   \@@_patch_booktabs:
1543   \box_clear_new:N \l_@@_cell_box
1544   \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}`).

```

1545 \bool_if:NT \l_@@_small_bool
1546 {
1547     \cs_set_nopar:Npn \arraystretch { 0.47 }
1548     \dim_set:Nn \arraycolsep { 1.45 pt }
1549 }

1550 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1551 {
1552     \tl_put_right:Nn \@@_begin_of_row:
1553     {
1554         \pgfsys@markposition
1555         { \@@_env: - row - \int_use:N \c@iRow - base }
1556     }
1557 }

```

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`.

```

1558 \cs_set_nopar:Npn \ialign
1559 {
1560     \@@_redefine_everycr:
1561     \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`.

```

1562 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1563 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1564 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1565 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1566 \dim_gzero_new:N \g_@@_ht_row_one_dim
1567 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1568 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1569 \dim_gzero_new:N \g_@@_ht_last_row_dim
1570 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1571 \dim_gzero_new:N \g_@@_dp_last_row_dim
1572 \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`.

```

1573 \cs_set_eq:NN \ialign \@@_old_ialign:
1574 \halign
1575 }

```

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).

```

1576 \cs_set_eq:NN \@@_old_ldots \ldots
1577 \cs_set_eq:NN \@@_old_cdots \cdots
1578 \cs_set_eq:NN \@@_old_vdots \vdots
1579 \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}`.

```

1580 \cs_set_eq:NN \@@_old_iddots \iddots
1581 \bool_if:NTF \l_@@_standard_cline_bool
1582 { \cs_set_eq:NN \cline \@@_standard_cline }
1583 { \cs_set_eq:NN \cline \@@_cline }
1584 \cs_set_eq:NN \Ldots \@@_Ldots
1585 \cs_set_eq:NN \Cdots \@@_Cdots
1586 \cs_set_eq:NN \Vdots \@@_Vdots
1587 \cs_set_eq:NN \Ddots \@@_Ddots
1588 \cs_set_eq:NN \Iddots \@@_Iddots
1589 \cs_set_eq:NN \Hline \@@_Hline:
1590 \cs_set_eq:NN \Hspace \@@_Hspace:
1591 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1592 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1593 \cs_set_eq:NN \Block \@@_Block:
1594 \cs_set_eq:NN \rotate \@@_rotate:
1595 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1596 \cs_set_eq:NN \dotfill \@@_dotfill:
1597 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1598 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1599 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1600 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1601 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1602 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1603 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1604 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1605 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1606 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1607 \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.

```

1608 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1609 \hook_gput_code:nnn { env / tabular / begin } { . }
1610 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }

```

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

```

1611 \tl_if_exist:NT \l_@@_note_in_caption_tl
1612 {
1613   \tl_if_empty:NF \l_@@_note_in_caption_tl
1614   {
1615     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1616     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1617   }
1618 }

```

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).

```

1619 \seq_gclear:N \g_@@_multicolumn_cells_seq
1620 \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).

```

1621 \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`).

```

1622 \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.

```

1623 \int_gzero_new:N \g_@@_col_total_int
1624 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1625 \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.

```

1626 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1627 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1628 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1629 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1630 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1631 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1632 \tl_gclear:N \g_nicematrix_code_before_tl
1633 \tl_gclear:N \g_@@_pre_code_before_tl
1634 }

```

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

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

```

1635 \cs_new_protected:Npn \@@_pre_array:
1636 {
1637   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1638   \int_gzero_new:N \c@iRow
1639   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1640   \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).

```

1641 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1642 {
1643   \bool_set_true:N \l_@@_last_row_without_value_bool
1644   \bool_if:NT \g_@@_aux_found_bool
1645     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1646 }
1647 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1648 {
1649   \bool_if:NT \g_@@_aux_found_bool
1650     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1651 }

```

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”.

```

1652 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1653 {
1654   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1655   {
1656     \dim_gset:Nn \g_@@_ht_last_row_dim
1657       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1658     \dim_gset:Nn \g_@@_dp_last_row_dim
1659       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1660   }
1661 }

```

```

1662 \seq_gclear:N \g_@@_cols_vlism_seq
1663 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1664 \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.

```

1665 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1666 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1667 \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.

```

1668 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `−2` is important.

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

```

1669 \@@_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.

```

1670 \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]`).

```

1671 \dim_zero_new:N \l_@@_left_delim_dim
1672 \dim_zero_new:N \l_@@_right_delim_dim
1673 \bool_if:NTF \g_@@_delims_bool
1674 {

```

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

```

1675 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1676 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1677 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1678 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1679 }
1680 {
1681 \dim_gset:Nn \l_@@_left_delim_dim
1682 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1683 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1684 }

```

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).

```

1685 \hbox_set:Nw \l_@@_the_array_box
1686 \skip_horizontal:N \l_@@_left_margin_dim
1687 \skip_horizontal:N \l_@@_extra_left_margin_dim
1688 \c_math_toggle_token
1689 \bool_if:NTF \l_@@_light_syntax_bool
1690 { \use:c { @@-light-syntax } }
1691 { \use:c { @@-normal-syntax } }
1692 }

```


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

```

1693 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1694 {
1695     \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1696     \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...

```

1697     \@@_pre_array:
1698 }

```

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).

```

1699 \cs_new_protected:Npn \@@_pre_code_before:
1700 {

```

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).

```

1701     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1702     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1703     \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1704     \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.

```

1705     \pgfsys@markposition { \@@_env: - position }
1706     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1707     \pgfpicture
1708     \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1709     \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1710     {
1711         \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1712         \pgfcoordinate { \@@_env: - row - ##1 }
1713         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1714     }

```

Now, the recreation of the `col` nodes.

```

1715     \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1716     {
1717         \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1718         \pgfcoordinate { \@@_env: - col - ##1 }
1719         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1720     }

```

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

```

1721     \@@_create_diag_nodes:

```

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

```

1722     \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1723     \endpgfpicture

```

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

```

1724 \@@_create_blocks_nodes:
1725 \IfPackageLoadedTF { tikz }
1726 {
1727   \tikzset
1728   {
1729     every~picture / .style =
1730     { overlay , name~prefix = \@@_env: - }
1731   }
1732 }
1733 { }
1734 \cs_set_eq:NN \cellcolor \@@_cellcolor
1735 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1736 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1737 \cs_set_eq:NN \rowcolor \@@_rowcolor
1738 \cs_set_eq:NN \rowcolors \@@_rowcolors
1739 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1740 \cs_set_eq:NN \arraycolor \@@_arraycolor
1741 \cs_set_eq:NN \columncolor \@@_columncolor
1742 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1743 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1744 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1745 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1746 }

```

```

1747 \cs_new_protected:Npn \@@_exec_code_before:
1748 {
1749   \seq_gclear_new:N \g_@@_colors_seq
1750   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1751   \group_begin:

```

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

```

1752   \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`).

```

1753   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1754   {
1755     \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1756     \@@_rescan_for_spanish:N \l_@@_code_before_tl
1757   }

```

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`.

```

1758   \exp_last_unbraced:NV \@@_CodeBefore_keys:
1759   \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.

```

1760   \@@_actually_color:
1761   \l_@@_code_before_tl
1762   \q_stop
1763   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1764   \group_end:
1765   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1766   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1767 }

```

```

1768 \keys_define:nn { NiceMatrix / CodeBefore }
1769 {
1770   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1771   create-cell-nodes .default:n = true ,
1772   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1773   sub-matrix .value_required:n = true ,
1774   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1775   delimiters / color .value_required:n = true ,
1776   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1777 }
1778 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1779 {
1780   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1781   \@@_CodeBefore:w
1782 }

```

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.

```

1783 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1784 {
1785   \bool_if:NT \g_@@_aux_found_bool
1786   {
1787     \@@_pre_code_before:
1788     #1
1789   }
1790 }

```

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.

```

1791 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1792 {
1793   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1794   {
1795     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1796     \pgfcoordinate { \@@_env: - row - ##1 - base }
1797     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1798     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1799     {
1800       \cs_if_exist:cT
1801       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1802       {
1803         \pgfsys@getposition
1804         { \@@_env: - ##1 - #####1 - NW }
1805         \@@_node_position:
1806         \pgfsys@getposition
1807         { \@@_env: - ##1 - #####1 - SE }
1808         \@@_node_position_i:
1809         \@@_pgf_rect_node:nnn
1810         { \@@_env: - ##1 - #####1 }
1811         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1812         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1813       }
1814     }
1815   }
1816   \int_step_inline:nn \c@iRow
1817   {
1818     \pgfnodealias
1819     { \@@_env: - ##1 - last }
1820     { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

1821     }
1822     \int_step_inline:nn \c@jCol
1823     {
1824         \pgfnodealias
1825         { \@@_env: - last - ##1 }
1826         { \@@_env: - \int_use:N \c@iRow - ##1 }
1827     }
1828     \@@_create_extra_nodes:
1829 }

```

```

1830 \cs_new_protected:Npn \@@_create_blocks_nodes:
1831 {
1832     \pgfpicture
1833     \pgf@relevantforpicturesizefalse
1834     \pgfrememberpicturepositiononpagetrue
1835     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1836     { \@@_create_one_block_node:nnnnn ##1 }
1837     \endpgfpicture
1838 }

```

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

```

1839 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1840 {
1841     \tl_if_empty:nF { #5 }
1842     {
1843         \@@_qpoint:n { col - #2 }
1844         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1845         \@@_qpoint:n { #1 }
1846         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1847         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1848         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1849         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1850         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1851         \@@_pgf_rect_node:nnnnn
1852         { \@@_env: - #5 }
1853         { \dim_use:N \l_tmpa_dim }
1854         { \dim_use:N \l_tmpb_dim }
1855         { \dim_use:N \l_@@_tmpc_dim }
1856         { \dim_use:N \l_@@_tmpd_dim }
1857     }
1858 }

```

```

1859 \cs_new_protected:Npn \@@_patch_for_revtext:
1860 {
1861     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1862     \cs_set_eq:NN \insert@column \insert@column@array
1863     \cs_set_eq:NN \@classx \@classx@array
1864     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1865     \cs_set_eq:NN \@arraycr \@arraycr@array
1866     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1867     \cs_set_eq:NN \array \array@array
1868     \cs_set_eq:NN \@array \@array@array
1869     \cs_set_eq:NN \@tabular \@tabular@array
1870     \cs_set_eq:NN \@mkpream \@mkpream@array
1871     \cs_set_eq:NN \endarray \endarray@array
1872     \cs_set:Npn \@tabarray { \ifnextchar [ { \array } { \array [ c ] } }
1873     \cs_set:Npn \endtabular { \endarray $\egroup} % $
1874 }

```

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).

11 The environment `{NiceArrayWithDelims}`

```

1875 \NewDocumentEnvironment { NiceArrayWithDelims }
1876 { m m 0 { } m ! 0 { } t \CodeBefore }
1877 {
1878   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1879   \@@_provide_pgfsyspdfmark:
1880   \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.

```

1881   \bgroup

1882   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1883   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1884   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1885   \int_gzero:N \g_@@_block_box_int
1886   \dim_zero:N \g_@@_width_last_col_dim
1887   \dim_zero:N \g_@@_width_first_col_dim
1888   \bool_gset_false:N \g_@@_row_of_col_done_bool
1889   \str_if_empty:NT \g_@@_name_env_str
1890   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1891   \bool_if:NTF \l_@@_tabular_bool
1892     \mode_leave_vertical:
1893     \@@_test_if_math_mode:
1894   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1895   \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.

```

1896   \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.

```

1897   \cs_if_exist:NT \tikz@library@external@loaded
1898   {
1899     \tikzexternaldisable
1900     \cs_if_exist:NT \ifstandalone
1901     { \tikzset { external / optimize = false } }
1902   }

```

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

```

1903   \int_gincr:N \g_@@_env_int
1904   \bool_if:NF \l_@@_block_auto_columns_width_bool
1905   { \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`).

```

1906   \seq_gclear:N \g_@@_blocks_seq
1907   \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`.

```

1908   \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1909   \seq_gclear:N \g_@@_pos_of_xdots_seq

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1910 \tl_gclear_new:N \g_@@_code_before_tl
1911 \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.

```

1912 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1913 {
1914     \bool_gset_true:N \g_@@_aux_found_bool
1915     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1916 }
1917 { \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.

```

1918 \tl_build_gbegin:N \g_@@_aux_tl
1919 \tl_if_empty:NF \g_@@_code_before_tl
1920 {
1921     \bool_set_true:N \l_@@_code_before_bool
1922     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1923 }
1924 \tl_if_empty:NF \g_@@_pre_code_before_tl
1925 { \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.

```

1926 \bool_if:NTF \g_@@_delims_bool
1927 { \keys_set:nn { NiceMatrix / pNiceArray } }
1928 { \keys_set:nn { NiceMatrix / NiceArray } }
1929 { #3 , #5 }

```

```

1930 \@@_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:.

```

1931 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1932 }

```

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

```

1933 {
1934     \bool_if:NTF \l_@@_light_syntax_bool
1935     { \use:c { end @@-light-syntax } }
1936     { \use:c { end @@-normal-syntax } }
1937     \c_math_toggle_token
1938     \skip_horizontal:N \l_@@_right_margin_dim
1939     \skip_horizontal:N \l_@@_extra_right_margin_dim
1940     \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.

```

1941 \bool_if:NT \l_@@_width_used_bool
1942 {
1943     \int_if_zero:nT \g_@@_total_X_weight_int
1944     { \@@_error_or_warning:n { width~without~X~columns } }
1945 }

```

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 .

```

1946 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1947 {
1948   \tl_build_gput_right:Nx \g_@@_aux_tl
1949   {
1950     \bool_set_true:N \l_@@_X_columns_aux_bool
1951     \dim_set:Nn \l_@@_X_columns_dim
1952     {
1953       \dim_compare:nNnTF
1954       {
1955         \dim_abs:n
1956         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1957       }
1958       <
1959       { 0.001 pt }
1960       { \dim_use:N \l_@@_X_columns_dim }
1961       {
1962         \dim_eval:n
1963         {
1964           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1965           / \int_use:N \g_@@_total_X_weight_int
1966           + \l_@@_X_columns_dim
1967         }
1968       }
1969     }
1970   }
1971 }

```

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

```

1972 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1973 {
1974   \bool_if:NF \l_@@_last_row_without_value_bool
1975   {
1976     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1977     {
1978       \@@_error:n { Wrong~last~row }
1979       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1980     }
1981   }
1982 }

```

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

```

1983 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1984 \bool_if:nTF \g_@@_last_col_found_bool
1985 { \int_gdecr:N \c@jCol }
1986 {
1987   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1988   { \@@_error:n { last~col~not~used } }
1989 }

```

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

```

1990 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1991 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 88).

```

1992 \int_if_zero:nT \l_@@_first_col_int
1993 {

```

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

```

1994         % \skip_horizontal:N \col@sep % 05-08-23
1995         \skip_horizontal:N \g_@@_width_first_col_dim
1996     }

```

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

```

1997     \bool_if:nTF { ! \g_@@_delims_bool }
1998     {
1999         \str_case:VnF \l_@@_baseline_tl
2000         {
2001             b \@@_use_arraybox_with_notes_b:
2002             c \@@_use_arraybox_with_notes_c:
2003         }
2004         \@@_use_arraybox_with_notes:
2005     }

```

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).

```

2006     {
2007         \int_if_zero:nTF \l_@@_first_row_int
2008         {
2009             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2010             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2011         }
2012         { \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

```

2013         \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2014         {
2015             \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2016             \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2017         }
2018         { \dim_zero:N \l_tmpb_dim }
2019         \hbox_set:Nn \l_tmpa_box
2020         {
2021             \c_math_toggle_token
2022             \@@_color:V \l_@@_delimiters_color_tl
2023             \exp_after:wN \left \g_@@_left_delim_tl
2024             \vcenter
2025             {

```

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.

```

2026             \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2027             \hbox
2028             {
2029                 \bool_if:nTF \l_@@_tabular_bool
2030                 { \skip_horizontal:N -\tabcolsep }
2031                 { \skip_horizontal:N -\arraycolsep }
2032                 \@@_use_arraybox_with_notes_c:
2033                 \bool_if:nTF \l_@@_tabular_bool
2034                 { \skip_horizontal:N -\tabcolsep }
2035                 { \skip_horizontal:N -\arraycolsep }
2036             }

```

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

```

2037             \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
2038         }

```

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.

```

2039             \@@_color:V \l_@@_delimiters_color_tl
2040             \exp_after:wN \right \g_@@_right_delim_tl

```

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).


```

2041         \c_math_toggle_token
2042     }

```

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.

```

2043     \bool_if:NTF \l_@@_delimiters_max_width_bool
2044     {
2045         \@@_put_box_in_flow_bis:nn
2046         \g_@@_left_delim_tl \g_@@_right_delim_tl
2047     }
2048     \@@_put_box_in_flow:
2049 }

```

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).

```

2050     \bool_if:NT \g_@@_last_col_found_bool
2051     {
2052         \skip_horizontal:N \g_@@_width_last_col_dim
2053         % \skip_horizontal:N \col@sep % 2023-08-05
2054     }
2055     \bool_if:NT \l_@@_preamble_bool
2056     {
2057         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2058         { \@@_warning_gredirect_none:n { columns-not-used } }
2059     }
2060     \@@_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.

```

2061     \egroup

```

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

```

2062     \tl_build_gend:N \g_@@_aux_tl
2063     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2064     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2065     \iow_now:Nx \@mainaux
2066     {
2067         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2068         { \exp_not:V \g_@@_aux_tl }
2069     }
2070     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2071     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2072 }

```

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.

```

2073     \cs_new_protected:Npn \@@_transform_preamble:
2074     {
2075         \@@_transform_preamble_i:
2076         \@@_transform_preamble_ii:
2077     }

```

```

2078 \cs_new_protected:Npn \@@_transform_preamble_i:
2079 {
2080   \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`).

```

2081   \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.

```

2082   \bool_gset_false:N \g_tmpb_bool

```

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

```

2083   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2084   \int_zero:N \l_tmpa_int
2085   \tl_gclear:N \g_@@_array_preamble_tl
2086   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2087   {
2088     \tl_gset:Nn \g_@@_array_preamble_tl
2089     { ! { \skip_horizontal:N \arrayrulewidth } }
2090   }
2091   {
2092     \clist_if_in:NnT \l_@@_vlines_clist 1
2093     {
2094       \tl_gset:Nn \g_@@_array_preamble_tl
2095       { ! { \skip_horizontal:N \arrayrulewidth } }
2096     }
2097   }

```

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

```

2098   \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2099   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2100   \@@_replace_columncolor:
2101 }

```

```

2102 \hook_gput_code:nnn { begindocument } { . }
2103 {
2104   \IfPackageLoadedTF { colortbl }
2105   {
2106     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2107     \cs_new_protected:Npn \@@_replace_columncolor:
2108     {
2109       \regex_replace_all:NnN
2110       \c_@@_columncolor_regex
2111       { \c { @@_columncolor_preamble } }
2112       \g_@@_array_preamble_tl
2113     }
2114   }
2115   {
2116     \cs_new_protected:Npn \@@_replace_columncolor:
2117     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2118   }
2119 }

```

```

2120 \cs_new_protected:Npn \@@_transform_preamble_ii:
2121 {

```

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

```

2122 \bool_lazy_or:nnT
2123   { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
2124   { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2125   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2126 \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).

```

2127 \int_if_zero:nTF \l_@@_first_col_int
2128 { \tl_gput_left:NV \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2129 {
2130   \bool_lazy_all:nT
2131   {
2132     { \bool_not_p:n \g_@@_delims_bool }
2133     { \bool_not_p:n \l_@@_tabular_bool }
2134     { \tl_if_empty_p:N \l_@@_vlines_clist }
2135     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2136   }
2137   { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2138 }
2139 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2140 { \tl_gput_right:NV \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2141 {
2142   \bool_lazy_all:nT
2143   {
2144     { \bool_not_p:n \g_@@_delims_bool }
2145     { \bool_not_p:n \l_@@_tabular_bool }
2146     { \tl_if_empty_p:N \l_@@_vlines_clist }
2147     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2148   }
2149   { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2150 }

```

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`).

```

2151 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2152 {
2153   \tl_gput_right:Nn \g_@@_array_preamble_tl
2154   { > { \@@_error_too_much_cols: } 1 }
2155 }
2156 }

```

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

```

2157 \cs_new_protected:Npn \@@_rec_preamble:n #1
2158 {

```

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

```

2159 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2160 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2161 {

```

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`.

Now, the columns defined by \newcolumntype of array.

```

2162     \cs_if_exist:cTF { NC @ find @ #1 }
2163     {
2164         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2165         \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2166     }
2167     {
2168         \tl_if_eq:nnT { #1 } { S }
2169         { \@@_fatal:n { unknown~column~type~S } }
2170         { \@@_fatal:nn { unknown~column~type } { #1 } }
2171     }
2172 }
2173 }

```

For c, l and r

```

2174 \cs_new:Npn \@@_c #1
2175 {
2176     \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2177     \tl_gclear:N \g_@@_pre_cell_tl
2178     \tl_gput_right:Nn \g_@@_array_preamble_tl
2179     {
2180         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2181         #1
2182         < \@@_cell_end:
2183     }

```

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

```

2184     \int_gincr:N \c@jCol
2185     \@@_rec_preamble_after_col:n
2186 }
2187 \cs_set_eq:NN \@@_l \@@_c
2188 \cs_set_eq:NN \@@_r \@@_c

```

For ! and @

```

2189 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2190 {
2191     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2192     \@@_rec_preamble:n
2193 }
2194 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2195 \cs_new:cpn { @@ _ | } #1
2196 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2197     \int_incr:N \l_tmpa_int
2198     \@@_make_preamble_i_i:n
2199 }
2200 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2201 {
2202     \str_if_eq:nnTF { #1 } |
2203     { \use:c { @@ _ | } | }
2204     { \@@_make_preamble_i_ii:nn { } #1 }
2205 }
2206 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2207 {
2208     \str_if_eq:nnTF { #2 } [
2209     { \@@_make_preamble_i_iii:nn { #1 } [ }
2210     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2211 ]
2212 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 [ #2 ]
2213 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

```

```

2214 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2215 {
2216   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2217   \tl_gput_right:Nx \g_@@_array_preamble_tl
2218   {

```

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

```

2219     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2220   }
2221   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2222   {
2223     \@@_vline:n
2224     {
2225       position = \int_eval:n { \c@jCol + 1 } ,
2226       multiplicity = \int_use:N \l_tmpa_int ,
2227       total-width = \dim_use:N \l_@@_rule_width_dim ,
2228       #2
2229     }

```

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.

```

2230   }
2231   \int_zero:N \l_tmpa_int
2232   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2233   \@@_rec_preamble:n #1
2234 }

```

```

2235 \cs_new:cpn { @@ _ > } #1 #2
2236 {
2237   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2238   \@@_rec_preamble:n
2239 }
2240 \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.

```

2241 \keys_define:nn { WithArrows / p-column }
2242 {
2243   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2244   r .value_forbidden:n = true ,
2245   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2246   c .value_forbidden:n = true ,
2247   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2248   l .value_forbidden:n = true ,
2249   R .code:n =
2250     \IfPackageLoadedTF { ragged2e }
2251     { \str_set:Nn \l_@@_hpos_col_str { R } }
2252     {
2253       \@@_error_or_warning:n { ragged2e-not-loaded }
2254       \str_set:Nn \l_@@_hpos_col_str { r }
2255     } ,
2256   R .value_forbidden:n = true ,
2257   L .code:n =
2258     \IfPackageLoadedTF { ragged2e }
2259     { \str_set:Nn \l_@@_hpos_col_str { L } }
2260     {
2261       \@@_error_or_warning:n { ragged2e-not-loaded }
2262       \str_set:Nn \l_@@_hpos_col_str { l }
2263     } ,
2264   L .value_forbidden:n = true ,
2265   C .code:n =
2266     \IfPackageLoadedTF { ragged2e }

```

```

2267     { \str_set:Nn \l_@@_hpos_col_str { C } }
2268     {
2269       \@@_error_or_warning:n { ragged2e-not-loaded }
2270       \str_set:Nn \l_@@_hpos_col_str { c }
2271     } ,
2272     C .value_forbidden:n = true ,
2273     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2274     S .value_forbidden:n = true ,
2275     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2276     p .value_forbidden:n = true ,
2277     t .meta:n = p ,
2278     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2279     m .value_forbidden:n = true ,
2280     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2281     b .value_forbidden:n = true ,
2282   }

```

For p, b and m.

```

2283 \cs_new:Npn \@@_p #1
2284 {
2285   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2286   \@@_make_preamble_ii_i:n
2287 }
2288 \cs_set_eq:NN \@@_b \@@_p
2289 \cs_set_eq:NN \@@_m \@@_p
2290 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2291 {
2292   \str_if_eq:nnTF { #1 } { [ ]
2293     { \@@_make_preamble_ii_ii:w [ ]
2294       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2295     }
2296   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2297     { \@@_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.

```

2298 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2299 {

```

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).

```

2300   \str_set:Nn \l_@@_hpos_col_str { j }
2301   \tl_set:Nn \l_tmpa_tl { #1 }
2302   \@@_keys_p_column:V \l_tmpa_tl
2303   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2304 }
2305 \cs_new_protected:Npn \@@_keys_p_column:n #1
2306 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2307 \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.

```

2308 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2309 {
2310   \use:e
2311   {
2312     \@@_make_preamble_ii_v:nnnnnnnn
2313     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2314     { \dim_eval:n { #1 } }
2315   }

```

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.

```

2316     \str_if_eq:VnTF \l_@@_hpos_col_str j
2317     { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2318     {
2319         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2320         { \str_lowercase:V \l_@@_hpos_col_str }
2321     }
2322     \str_case:Vn \l_@@_hpos_col_str
2323     {
2324         c { \exp_not:N \centering }
2325         l { \exp_not:N \raggedright }
2326         r { \exp_not:N \raggedleft }
2327         C { \exp_not:N \Centering }
2328         L { \exp_not:N \RaggedRight }
2329         R { \exp_not:N \RaggedLeft }
2330     }
2331     #3
2332 }
2333 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2334 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2335 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2336 { #2 }
2337 {
2338     \str_case:VnF \l_@@_hpos_col_str
2339     {
2340         { j } { c }
2341         { si } { c }
2342     }

```

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

```

2343         { \str_lowercase:V \l_@@_hpos_col_str }
2344     }
2345 }

```

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

```

2346     \int_gincr:N \c@jCol
2347     \@@_rec_preamble_after_col:n
2348 }

```

#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*.

```

2349 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2350 {
2351     \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2352     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2353     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2354     \tl_gput_right:Nv \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2355     \tl_gclear:N \g_@@_pre_cell_tl
2356     \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2357 {
2358   > {

```

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.

```

2359       \dim_set:Nn \l_@@_col_width_dim { #2 }
2360       \@@_cell_begin:w

```

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

```

2361       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2362       \everypar
2363       {
2364         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2365         \everypar { }
2366       }

```

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

```

2367       #3

```

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

```

2368       \g_@@_row_style_tl
2369       \arraybackslash
2370       #5
2371     }
2372     #8
2373     < {
2374       #6

```

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

```

2375       \@finalstrut \@arstrutbox
2376       \use:c { end #7 }

```

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

```

2377       #4
2378       \@@_cell_end:
2379     }
2380   }
2381 }

```

```

2382 \str_new:N \c_@@_ignorespaces_str
2383 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2384 \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...

```

2385 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2386 \cs_new_protected:Npn \@@_test_if_empty_i:
2387 {
2388   \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2389   \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2390   { \@@_test_if_empty:w }
2391 }
2392 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2393 {
2394   \peek_meaning:NT \unskip
2395   {
2396     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2397     {

```



```

2398         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2399         \skip_horizontal:N \l_@@_col_width_dim
2400     }
2401 }
2402 }
2403 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2404 {
2405     \peek_meaning:NT \__siunitx_table_skip:n
2406     {
2407         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2408         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2409     }
2410 }

```

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.

```

2411 \cs_new_protected:Npn \@@_center_cell_box:
2412 {

```

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

```

2413     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2414     {
2415         \int_compare:nNnT
2416         { \box_ht:N \l_@@_cell_box }
2417         >

```

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).

```

2418         { \box_ht:N \strutbox }
2419     {
2420         \hbox_set:Nn \l_@@_cell_box
2421         {
2422             \box_move_down:nn
2423             {
2424                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2425                 + \baselineskip ) / 2
2426             }
2427             { \box_use:N \l_@@_cell_box }
2428         }
2429     }
2430 }
2431 }

```

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

```

2432 \cs_new:Npn \@@_V #1 #2
2433 {
2434     \str_if_eq:nnTF { #2 } { [ ] }
2435     { \@@_make_preamble_V_i:w [ ] }
2436     { \@@_make_preamble_V_i:w [ ] { #2 } }
2437 }
2438 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2439 { \@@_make_preamble_V_ii:nn { #1 } }
2440 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2441 {
2442     \str_set:Nn \l_@@_vpos_col_str { p }
2443     \str_set:Nn \l_@@_hpos_col_str { j }
2444     \tl_set:Nn \l_tmpa_tl { #1 }

```

```

2445 \@@_keys_p_column:V \l_tmpa_tl
2446 \IfPackageLoadedTF { varwidth }
2447 { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2448 {
2449 \@@_error_or_warning:n { varwidth~not~loaded }
2450 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2451 }
2452 }

```

For *w* and *W*

```

2453 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2454 \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.
2455 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2456 {
2457 \str_if_eq:nnTF { #3 } { s }
2458 { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2459 { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2460 }

```

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.

```

2461 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2462 {
2463 \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2464 \tl_gclear:N \g_@@_pre_cell_tl
2465 \tl_gput_right:Nn \g_@@_array_preamble_tl
2466 {
2467 > {
2468 \dim_set:Nn \l_@@_col_width_dim { #2 }
2469 \@@_cell_begin:w
2470 \str_set:Nn \l_@@_hpos_cell_str { c }
2471 }
2472 c
2473 < {
2474 \@@_cell_end_for_w_s:
2475 #1
2476 \@@_adjust_size_box:
2477 \box_use_drop:N \l_@@_cell_box
2478 }
2479 }
2480 \int_gincr:N \c@jCol
2481 \@@_rec_preamble_after_col:n
2482 }

```

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

```

2483 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2484 {
2485 \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2486 \tl_gclear:N \g_@@_pre_cell_tl
2487 \tl_gput_right:Nn \g_@@_array_preamble_tl
2488 {
2489 > {

```

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.

```

2490         \dim_set:Nn \l_@@_col_width_dim { #4 }
2491         \hbox_set:Nw \l_@@_cell_box
2492         \@@_cell_begin:w
2493         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2494     }
2495     c
2496     < {
2497         \@@_cell_end:
2498         \hbox_set_end:
2499         #1
2500         \@@_adjust_size_box:
2501         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2502     }
2503 }

```

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

```

2504     \int_gincr:N \c@jCol
2505     \@@_rec_preamble_after_col:n
2506 }

```

```

2507 \cs_new_protected:Npn \@@_special_W:
2508 {
2509     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2510     { \@@_warning:n { W~warning } }
2511 }

```

For `S` (of `siunitx`).

```

2512 \cs_new:Npn \@@_S #1 #2
2513 {
2514     \str_if_eq:nnTF { #2 } { [ ] }
2515     { \@@_make_preamble_S:w [ ] }
2516     { \@@_make_preamble_S:w [ ] { #2 } }
2517 }
2518 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2519 { \@@_make_preamble_S_i:n { #1 } }
2520 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2521 {
2522     \IfPackageLoadedTF { siunitx }
2523     {
2524         \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2525         \tl_gclear:N \g_@@_pre_cell_tl
2526         \tl_gput_right:Nn \g_@@_array_preamble_tl
2527         {
2528             > {
2529                 \@@_cell_begin:w
2530                 \keys_set:nn { siunitx } { #1 }
2531                 \siunitx_cell_begin:w
2532             }
2533             c
2534             < { \siunitx_cell_end: \@@_cell_end: }
2535         }

```

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

```

2536     \int_gincr:N \c@jCol
2537     \@@_rec_preamble_after_col:n
2538 }
2539 { \@@_fatal:n { siunitx~not~loaded } }
2540 }

```

For (, [and \{.

```

2541 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2542 {
2543   \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.

```

2544   \int_if_zero:nTF \c@jCol
2545   {
2546     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2547     {

```

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

```

2548       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2549       \tl_gset:Nn \g_@@_right_delim_tl { . }
2550       \@@_rec_preamble:n #2
2551     }
2552   {
2553     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2554     \@@_make_preamble_iv:nn { #1 } { #2 }
2555   }
2556 }
2557 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2558 }
2559 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2560 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2561 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2562 {
2563   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2564   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }
2565   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2566   {
2567     \@@_error:nn { delimiter~after~opening } { #2 }
2568     \@@_rec_preamble:n
2569   }
2570   { \@@_rec_preamble:n #2 }
2571 }

```

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

```

2572 \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 a 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}).

```

2573 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2574 {
2575   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2576   \tl_if_in:nnTF { ) ] \} } { #2 }
2577   { \@@_make_preamble_v:nnn #1 #2 }
2578   {
2579     \tl_if_eq:nnTF { \stop } { #2 }
2580     {
2581       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2582       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2583       {
2584         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2585         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2586         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } } \c_false_bool }
2587       \@@_rec_preamble:n #2
2588     }
2589   }
2590   {

```

```

2591 \tl_if_in:nnT { ( [ \{ \left } { #2 }
2592 { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2593 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2594 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2595 \@@_rec_preamble:n #2
2596 }
2597 }
2598 }
2599 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ] }
2600 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2601 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2602 {
2603   \tl_if_eq:nnTF { \stop } { #3 }
2604   {
2605     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2606     {
2607       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2608       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2609       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2610       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2611     }
2612     {
2613       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2614       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2615       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2616       \@@_error:nn { double~closing~delimiter } { #2 }
2617     }
2618   }
2619   {
2620     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2621     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2622     \@@_error:nn { double~closing~delimiter } { #2 }
2623     \@@_rec_preamble:n #3
2624   }
2625 }
2626 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2627 { \use:c { @@ _ \token_to_str:N ) } }

```

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

```

2628 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2629 {
2630   \str_if_eq:nnTF { #1 } { < }
2631   \@@_rec_preamble_after_col_i:n
2632   {
2633     \str_if_eq:nnTF { #1 } { @ }
2634     \@@_rec_preamble_after_col_ii:n
2635     {
2636       \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2637       {
2638         \tl_gput_right:Nn \g_@@_array_preamble_tl
2639         { ! { \skip_horizontal:N \arrayrulewidth } }
2640       }
2641       {
2642         \exp_args:NNe
2643         \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2644         {
2645           \tl_gput_right:Nn \g_@@_array_preamble_tl
2646           { ! { \skip_horizontal:N \arrayrulewidth } }
2647         }
2648       }
2649     }
2650   }
2651 }

```

```

2648         }
2649         \@@_rec_preamble:n { #1 }
2650     }
2651 }
2652 }
2653 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2654 {
2655     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2656     \@@_rec_preamble_after_col:n
2657 }

```

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.

```

2658 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2659 {
2660     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2661     {
2662         \tl_gput_right:Nn \g_@@_array_preamble_tl
2663         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2664     }
2665     {
2666         \exp_args:NNe
2667         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2668         {
2669             \tl_gput_right:Nn \g_@@_array_preamble_tl
2670             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2671         }
2672         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2673     }
2674     \@@_rec_preamble:n
2675 }

2676 \cs_new:cpn { @@ _ * } #1 #2 #3
2677 {
2678     \tl_clear:N \l_tmpa_tl
2679     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2680     \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpa_tl
2681 }

```

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

```

2682 \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`.

```

2683 \cs_new:Npn \@@_X #1 #2
2684 {
2685     \str_if_eq:nnTF { #2 } { [ ]
2686     { \@@_make_preamble_X:w [ ]
2687     { \@@_make_preamble_X:w [ ] #2 }
2688     }
2689 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2690 { \@@_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`).

```

2691 \keys_define:nn { WithArrows / X-column }
2692 { 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.

```
2693 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2694 {
```

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).

```
2695 \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).

```
2696 \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.

```
2697 \int_zero_new:N \l_@@_weight_int
2698 \int_set:Nn \l_@@_weight_int { 1 }
2699 \tl_set:Nn \l_tmpa_tl { #1 }
2700 \@@_keys_p_column:V \l_tmpa_tl
2701 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2702 \int_compare:nNnT \l_@@_weight_int < 0
2703 {
2704   \@@_error_or_warning:n { negative-weight }
2705   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2706 }
2707 \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).

```
2708 \bool_if:NTF \l_@@_X_columns_aux_bool
2709 {
2710   \exp_args:Nne
2711   \@@_make_preamble_ii_iv:nnn
2712   { \l_@@_weight_int \l_@@_X_columns_dim }
2713   { minipage }
2714   { \@@_no_update_width: }
2715 }
2716 {
2717   \tl_gput_right:Nn \g_@@_array_preamble_tl
2718   {
2719     > {
2720       \@@_cell_begin:w
2721       \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.

```
2722 \NotEmpty
```

The following code will nullify the box of the cell.

```
2723 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2724 { \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.

```
2725 \begin { minipage } { 5 cm } \arraybackslash
2726 }
2727 c
2728 < {
2729   \end { minipage }
2730   \@@_cell_end:
2731 }
2732 }
```

```

2733     \int_gincr:N \c@jCol
2734     \@@_rec_preamble_after_col:n
2735   }
2736 }

2737 \cs_new_protected:Npn \@@_no_update_width:
2738 {
2739   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2740   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2741 }

```

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

```

2742 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2743 {
2744   \seq_gput_right:Nx \g_@@_cols_vlism_seq
2745   { \int_eval:n { \c@jCol + 1 } }
2746   \tl_gput_right:Nx \g_@@_array_preamble_tl
2747   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2748   \@@_rec_preamble:n
2749 }

```

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.

```

2750 \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).

```

2751 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2752 { \@@_fatal:n { Preamble~forgotten } }
2753 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2754 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2755 \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).

```

2756 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2757 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2758   \multispan { #1 }
2759   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2760   \begingroup
2761   \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

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

```

2762   \tl_gclear:N \g_@@_preamble_tl
2763   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2764   \exp_args:NV \mkpream \g_@@_preamble_tl
2765   \@addtopreamble \@empty
2766   \endgroup

```


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

```

2767 \int_compare:nNnT { #1 } > 1
2768 {
2769   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2770   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2771   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2772   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2773   {
2774     {
2775       \int_if_zero:nTF \c@jCol
2776       { \int_eval:n { \c@iRow + 1 } }
2777       { \int_use:N \c@iRow }
2778     }
2779     { \int_eval:n { \c@jCol + 1 } }
2780     {
2781       \int_if_zero:nTF \c@jCol
2782       { \int_eval:n { \c@iRow + 1 } }
2783       { \int_use:N \c@iRow }
2784     }
2785     { \int_eval:n { \c@jCol + #1 } }
2786     { } % for the name of the block
2787   }
2788 }

```

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

```

2789 \cs_set:Npn \@sharp { #3 }
2790 \@arstrut
2791 \@preamble
2792 \null

```

We add some lines.

```

2793 \int_gadd:Nn \c@jCol { #1 - 1 }
2794 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2795 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2796 \ignorespaces
2797 }

```

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`.

```

2798 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2799 {
2800   \str_case:nnF { #1 }
2801   {
2802     c { \@@_make_m_preamble_i:n #1 }
2803     l { \@@_make_m_preamble_i:n #1 }
2804     r { \@@_make_m_preamble_i:n #1 }
2805     > { \@@_make_m_preamble_ii:nn #1 }
2806     ! { \@@_make_m_preamble_ii:nn #1 }
2807     @ { \@@_make_m_preamble_ii:nn #1 }
2808     | { \@@_make_m_preamble_iii:n #1 }
2809     p { \@@_make_m_preamble_iv:nnn t #1 }
2810     m { \@@_make_m_preamble_iv:nnn c #1 }
2811     b { \@@_make_m_preamble_iv:nnn b #1 }
2812     w { \@@_make_m_preamble_v:nnnn { } #1 }
2813     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2814     \q_stop { }
2815   }
2816   {
2817     \cs_if_exist:cTF { NC @ find @ #1 }
2818     {
2819       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }

```

```

2820         \exp_last_unbraced:NV \@@_make_m_preamble:n \l_tmpa_tl
2821     }
2822     {
2823         \tl_if_eq:nnT { #1 } { S }
2824         { \@@_fatal:n { unknown~column~type~S } }
2825         { \@@_fatal:nn { unknown~column~type } { #1 } }
2826     }
2827 }
2828 }

```

For c, l and r

```

2829 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2830 {
2831     \tl_gput_right:Nn \g_@@_preamble_tl
2832     {
2833         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2834         #1
2835         < \@@_cell_end:
2836     }

```

We test for the presence of a <.

```

2837     \@@_make_m_preamble_x:n
2838 }

```

For >, ! and @

```

2839 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2840 {
2841     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2842     \@@_make_m_preamble:n
2843 }

```

For l

```

2844 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2845 {
2846     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2847     \@@_make_m_preamble:n
2848 }

```

For p, m and b

```

2849 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2850 {
2851     \tl_gput_right:Nn \g_@@_preamble_tl
2852     {
2853         > {
2854             \@@_cell_begin:w
2855             \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2856             \mode_leave_vertical:
2857             \arraybackslash
2858             \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2859         }
2860         c
2861         < {
2862             \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2863             \end { minipage }
2864             \@@_cell_end:
2865         }
2866     }

```

We test for the presence of a <.

```

2867     \@@_make_m_preamble_x:n
2868 }

```

For w and W

```

2869 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2870 {
2871   \tl_gput_right:Nn \g_@@_preamble_tl
2872   {
2873     > {
2874       \dim_set:Nn \l_@@_col_width_dim { #4 }
2875       \hbox_set:Nw \l_@@_cell_box
2876       \@@_cell_begin:w
2877       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2878     }
2879     c
2880     < {
2881       \@@_cell_end:
2882       \hbox_set_end:
2883       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2884       #1
2885       \@@_adjust_size_box:
2886       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2887     }
2888   }

```

We test for the presence of a <.

```

2889   \@@_make_m_preamble_x:n
2890 }

```

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

```

2891 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2892 {
2893   \str_if_eq:nnTF { #1 } { < }
2894   \@@_make_m_preamble_ix:n
2895   { \@@_make_m_preamble:n { #1 } }
2896 }
2897 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2898 {
2899   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2900   \@@_make_m_preamble_x:n
2901 }

```

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).

```

2902 \cs_new_protected:Npn \@@_put_box_in_flow:
2903 {
2904   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2905   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2906   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2907   { \box_use_drop:N \l_tmpa_box }
2908   \@@_put_box_in_flow_i:
2909 }

```

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).

```

2910 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2911 {
2912   \pgfpicture
2913   \@@_qpoint:n { row - 1 }
2914   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2915   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2916   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2917   \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).

```

2918 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2919 {
2920   \int_set:Nn \l_tmpa_int
2921   {
2922     \str_range:Nnn
2923     \l_@@_baseline_tl
2924     6
2925     { \tl_count:V \l_@@_baseline_tl }
2926   }
2927   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2928 }
2929 {
2930   \str_case:VnF \l_@@_baseline_tl
2931   {
2932     { t } { \int_set:Nn \l_tmpa_int 1 }
2933     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2934   }
2935   { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2936   \bool_lazy_or:nnT
2937   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2938   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2939   {
2940     \@@_error:n { bad~value~for~baseline }
2941     \int_set:Nn \l_tmpa_int 1
2942   }
2943   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2944 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2945 }
2946 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2947 \endpgfpicture
2948 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2949 \box_use_drop:N \l_tmpa_box
2950 }

```

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).

```

2951 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2952 {

```

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.

```

2953 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2954 {
2955   \int_compare:nNnT \c@jCol > 1 % added 2023-08-13
2956   {
2957     \box_set_wd:Nn \l_@@_the_array_box
2958     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2959   }
2960 }

```

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

```

2961 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2962 \bool_if:NT \l_@@_caption_above_bool
2963 {
2964   \tl_if_empty:NF \l_@@_caption_tl

```

```

2965     {
2966         \bool_set_false:N \g_@@_caption_finished_bool
2967         \int_gzero:N \c@tabularnote
2968         \@@_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).

```

2969         \int_compare:nNnT \g_@@_notes_caption_int > 0
2970         {
2971             \tl_build_gput_right:Nx \g_@@_aux_tl
2972             {
2973                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2974                 { \int_use:N \g_@@_notes_caption_int }
2975             }
2976             \int_gzero:N \g_@@_notes_caption_int
2977         }
2978     }
2979 }

```

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

```

2980     \hbox
2981     {
2982         \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.

```

2983         \@@_create_extra_nodes:
2984         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2985     }

```

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

```

2986     \bool_lazy_any:nT
2987     {
2988         { ! \seq_if_empty_p:N \g_@@_notes_seq }
2989         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2990         { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2991     }
2992     \@@_insert_tabularnotes:
2993     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2994     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2995     \end { minipage }
2996 }

```

```

2997 \cs_new_protected:Npn \@@_insert_caption:
2998 {
2999     \tl_if_empty:NF \l_@@_caption_tl
3000     {
3001         \cs_if_exist:NTF \@capytype
3002         { \@@_insert_caption_i: }
3003         { \@@_error:n { caption-outside-float } }
3004     }
3005 }

```

```

3006 \cs_new_protected:Npn \@@_insert_caption_i:
3007 {
3008     \group_begin:

```

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

```
3009 \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.

```
3010 \IfPackageLoadedTF { floatrow }
3011 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3012 { }
3013 \tl_if_empty:NTF \l_@@_short_caption_tl
3014 { \caption }
3015 { \caption [ \l_@@_short_caption_tl ] }
3016 { \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.

```
3017 \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
3018 {
3019   \bool_gset_true:N \g_@@_caption_finished_bool
3020   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3021   \int_gzero:N \c@tabularnote
3022 }
3023 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3024 \group_end:
3025 }

3026 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3027 {
3028   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3029   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3030 }

3031 \cs_new_protected:Npn \@@_insert_tabularnotes:
3032 {
3033   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3034   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3035   \skip_vertical:N 0.65ex
```

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

```
3036 \group_begin:
3037 \l_@@_notes_code_before_tl
3038 \tl_if_empty:NF \g_@@_tabularnote_tl
3039 {
3040   \g_@@_tabularnote_tl \par
3041   \tl_gclear:N \g_@@_tabularnote_tl
3042 }
```

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.

```
3043 \int_compare:nNnT \c@tabularnote > 0
3044 {
3045   \bool_if:NTF \l_@@_notes_para_bool
3046   {
3047     \begin { tabularnotes* }
3048     \seq_map_inline:Nn \g_@@_notes_seq
3049     { \@@_one_tabularnote:nn ##1 }
3050     \strut
3051     \end { tabularnotes* }
```

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

```

3052         \par
3053     }
3054     {
3055         \tabularnotes
3056         \seq_map_inline:Nn \g_@@_notes_seq
3057         { \@@_one_tabularnote:nn ##1 }
3058         \strut
3059         \endtabularnotes
3060     }
3061 }
3062 \unskip
3063 \group_end:
3064 \bool_if:NT \l_@@_notes_bottomrule_bool
3065 {
3066     \IfPackageLoadedTF { booktabs }
3067     {

```

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

```

3068         \skip_vertical:N \aboverulesep

```

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

```

3069         { \CT@arc@ \hrule height \heavyrulewidth }
3070     }
3071     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3072 }
3073 \l_@@_notes_code_after_tl
3074 \seq_gclear:N \g_@@_notes_seq
3075 \seq_gclear:N \g_@@_notes_in_caption_seq
3076 \int_gzero:N \c@tabularnote
3077 }

```

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.

```

3078 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3079 {
3080     \tl_if_novalue:nTF { #1 }
3081     { \item }
3082     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3083 }

```

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`.

```

3084 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3085 {
3086     \pgfpicture
3087     \@@_qpoint:n { row - 1 }
3088     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3089     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3090     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3091     \endpgfpicture
3092     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3093     \int_if_zero:nT \l_@@_first_row_int
3094     {
3095         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3096         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3097     }
3098     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3099 }

```

Now, the general case.

```

3100 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3101 {

```

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

```

3102 \tl_if_eq:NnT \l_@@_baseline_tl { t }
3103 { \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`.

```

3104 \pgfpicture
3105 \@@_qpoint:n { row - 1 }
3106 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3107 \str_if_in:NnTF \l_@@_baseline_tl { line- }
3108 {
3109   \int_set:Nn \l_tmpa_int
3110   {
3111     \str_range:Nnn
3112     \l_@@_baseline_tl
3113     6
3114     { \tl_count:V \l_@@_baseline_tl }
3115   }
3116   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3117 }
3118 {
3119   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3120   \bool_lazy_or:nnT
3121   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3122   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3123   {
3124     \@@_error:n { bad-value-for-baseline }
3125     \int_set:Nn \l_tmpa_int 1
3126   }
3127   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3128 }
3129 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3130 \endpgfpicture
3131 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3132 \int_if_zero:nT \l_@@_first_row_int
3133 {
3134   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3135   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3136 }
3137 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3138 }

```

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.

```

3139 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3140 {

```

We will compute the real width of both delimiters used.

```

3141 \dim_zero_new:N \l_@@_real_left_delim_dim
3142 \dim_zero_new:N \l_@@_real_right_delim_dim
3143 \hbox_set:Nn \l_tmpb_box
3144 {
3145   \c_math_toggle_token
3146   \left #1
3147   \vcenter
3148   {
3149     \vbox_to_ht:nn
3150     { \box_ht_plus_dp:N \l_tmpa_box }
3151     { }

```



```

3152     }
3153     \right .
3154     \c_math_toggle_token
3155   }
3156   \dim_set:Nn \l_@@_real_left_delim_dim
3157   { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
3158   \hbox_set:Nn \l_tmpb_box
3159   {
3160     \c_math_toggle_token
3161     \left .
3162     \vbox_to_ht:nn
3163     { \box_ht_plus_dp:N \l_tmpa_box }
3164     { }
3165     \right #2
3166     \c_math_toggle_token
3167   }
3168   \dim_set:Nn \l_@@_real_right_delim_dim
3169   { \box_wd:N \l_tmpb_box - \nullldelimiterspace }

```

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

```

3170   \skip_horizontal:N \l_@@_left_delim_dim
3171   \skip_horizontal:N -\l_@@_real_left_delim_dim
3172   \@@_put_box_in_flow:
3173   \skip_horizontal:N \l_@@_right_delim_dim
3174   \skip_horizontal:N -\l_@@_real_right_delim_dim
3175 }

```

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).

```

3176 \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).

```

3177 {
3178   \peek_remove_spaces:n
3179   {
3180     \peek_meaning:NTF \end
3181     \@@_analyze_end:Nn
3182     {
3183       \@@_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`).

```

3184       \exp_args:NV \@@_array: \g_@@_array_preamble_tl
3185     }
3186   }
3187 }
3188 {
3189   \@@_create_col_nodes:
3190   \endarray
3191 }

```

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

```

3192 \NewDocumentEnvironment { @@-light-syntax } { b }
3193 {

```

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.

```

3194 \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3195 \tl_map_inline:nn { #1 }
3196 {
3197   \str_if_eq:nnT { ##1 } { & }
3198   { \@@_fatal:n { ampersand-in-light-syntax } }
3199   \str_if_eq:nnT { ##1 } { \ }
3200   { \@@_fatal:n { double-backslash-in-light-syntax } }
3201 }

```

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`.

```

3202 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3203 }

```

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.

```

3204 {
3205   \@@_create_col_nodes:
3206   \endarray
3207 }
3208 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3209 {
3210   \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).

```

3211 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3212 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3213 \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3214 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3215 \tl_if_empty:NF \l_tmpa_tl
3216 { \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.

```

3217 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3218 { \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`).

```

3219 \tl_build_begin:N \l_@@_new_body_tl
3220 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3221 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3222 \@@_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).

```

3223 \seq_map_inline:Nn \l_@@_rows_seq
3224 {
3225   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3226   \@@_line_with_light_syntax:n { ##1 }
3227 }
3228 \tl_build_end:N \l_@@_new_body_tl
3229 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3230 {
3231   \int_set:Nn \l_@@_last_col_int
3232     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3233 }

```

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.

```

3234 \@@_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`).

```

3235 \exp_args:NV \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3236 }
3237 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3238 {
3239   \seq_clear_new:N \l_@@_cells_seq
3240   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3241   \int_set:Nn \l_@@_nb_cols_int
3242     {
3243       \int_max:nn
3244         \l_@@_nb_cols_int
3245         { \seq_count:N \l_@@_cells_seq }
3246     }
3247   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3248   \exp_args:NNV \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3249   \seq_map_inline:Nn \l_@@_cells_seq
3250     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3251 }
3252 \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`.

```

3253 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3254 {
3255   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3256   { \@@_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.

```

3257   \end { #2 }
3258 }

```

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).

```

3259 \cs_new:Npn \@@_create_col_nodes:
3260 {
3261   \crrcr
3262   \int_if_zero:nT \l_@@_first_col_int
3263     {
3264       \omit

```

```

3265 \hbox_overlap_left:n
3266 {
3267   \bool_if:NT \l_@@_code_before_bool
3268     { \pgfsys@markposition { \@@_env: - col - 0 } }
3269   \pgfpicture
3270   \pgfrememberpicturerepositiononpagetrue
3271   \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3272   \str_if_empty:NF \l_@@_name_str
3273     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3274   \endpgfpicture
3275   \skip_horizontal:N 2\col@sep
3276   \skip_horizontal:N \g_@@_width_first_col_dim
3277 }
3278 &
3279 }
3280 \omit

```

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

```

3281 \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`).

```

3282 \int_if_zero:nTF \l_@@_first_col_int
3283 {
3284   \bool_if:NT \l_@@_code_before_bool
3285   {
3286     \hbox
3287     {
3288       \skip_horizontal:N -0.5\arrayrulewidth
3289       \pgfsys@markposition { \@@_env: - col - 1 }
3290       \skip_horizontal:N 0.5\arrayrulewidth
3291     }
3292   }
3293   \pgfpicture
3294   \pgfrememberpicturerepositiononpagetrue
3295   \pgfcoordinate { \@@_env: - col - 1 }
3296     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3297   \str_if_empty:NF \l_@@_name_str
3298     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3299   \endpgfpicture
3300 }
3301 {
3302   \bool_if:NT \l_@@_code_before_bool
3303   {
3304     \hbox
3305     {
3306       \skip_horizontal:N 0.5\arrayrulewidth
3307       \pgfsys@markposition { \@@_env: - col - 1 }
3308       \skip_horizontal:N -0.5\arrayrulewidth
3309     }
3310   }
3311   \pgfpicture
3312   \pgfrememberpicturerepositiononpagetrue
3313   \pgfcoordinate { \@@_env: - col - 1 }
3314     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3315   \str_if_empty:NF \l_@@_name_str
3316     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3317   \endpgfpicture
3318 }

```

We compute in `\g_tmpa_skip` the common width of the columns (it's a skip and not a dimension). We use a global variable because we are in a cell of an `\halign` and because we have to use that variable in other cells (of the same row). The affectation of `\g_tmpa_skip`, like all the affectations, must be done after the `\omit` of the cell.

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but we will add some dimensions to it.

```

3319 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3320 \bool_if:NF \l_@@_auto_columns_width_bool
3321 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3322 {
3323   \bool_lazy_and:nnTF
3324     \l_@@_auto_columns_width_bool
3325     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3326     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3327     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3328     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3329 }
3330 \skip_horizontal:N \g_tmpa_skip
3331 \hbox
3332 {
3333   \bool_if:NT \l_@@_code_before_bool
3334   {
3335     \hbox
3336     {
3337       \skip_horizontal:N -0.5\arrayrulewidth
3338       \pgfsys@markposition { \@@_env: - col - 2 }
3339       \skip_horizontal:N 0.5\arrayrulewidth
3340     }
3341   }
3342   \pgfpicture
3343   \pgfrememberpicturerepositiononpagetrue
3344   \pgfcoordinate { \@@_env: - col - 2 }
3345   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3346   \str_if_empty:NF \l_@@_name_str
3347   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3348   \endpgfpicture
3349 }

```

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.

```

3350 \int_gset:Nn \g_tmpa_int 1
3351 \bool_if:NTF \g_@@_last_col_found_bool
3352 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3353 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3354 {
3355   &
3356   \omit
3357   \int_gincr:N \g_tmpa_int

```

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

```

3358   \skip_horizontal:N \g_tmpa_skip
3359   \bool_if:NT \l_@@_code_before_bool
3360   {
3361     \hbox
3362     {
3363       \skip_horizontal:N -0.5\arrayrulewidth
3364       \pgfsys@markposition
3365       { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3366       \skip_horizontal:N 0.5\arrayrulewidth
3367     }
3368   }

```

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

```

3369   \pgfpicture
3370   \pgfrememberpicturerepositiononpagetrue
3371   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3372   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3373   \str_if_empty:NF \l_@@_name_str

```

```

3374     {
3375         \pgfnodealias
3376         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3377         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3378     }
3379 \endpgfpicture
3380 }

```

```

3381 &
3382 \omit

```

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

```

3383 \int_compare:nNt \g_@@_col_total_int = 1
3384 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3385 \skip_horizontal:N \g_tmpa_skip
3386 \int_gincr:N \g_tmpa_int
3387 \bool_lazy_all:nT
3388 {
3389     { \bool_not_p:n \g_@@_delims_bool }
3390     { \bool_not_p:n \l_@@_tabular_bool }
3391     { \clist_if_empty_p:N \l_@@_vlines_clist }
3392     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3393     { ! \l_@@_bar_at_end_of_pream_bool }
3394 }
3395 { \skip_horizontal:N -\col@sep }
3396 \bool_if:NT \l_@@_code_before_bool
3397 {
3398     \hbox
3399     {
3400         \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.

```

3401         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3402         { \skip_horizontal:N -\arraycolsep }
3403         \pgfsys@markposition
3404         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3405         \skip_horizontal:N 0.5\arrayrulewidth
3406         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3407         { \skip_horizontal:N \arraycolsep }
3408     }
3409 }
3410 \pgfpicture
3411 \pgfrememberpicturepositiononpagetrue
3412 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3413 {
3414     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3415     {
3416         \pgfpoint
3417         { - 0.5 \arrayrulewidth - \arraycolsep }
3418         \c_zero_dim
3419     }
3420     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3421 }
3422 \str_if_empty:NF \l_@@_name_str
3423 {
3424     \pgfnodealias
3425     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3426     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3427 }
3428 \endpgfpicture

```

```

3429 \bool_if:NT \g_@@_last_col_found_bool
3430 {
3431   \hbox_overlap_right:n
3432   {
3433     \skip_horizontal:N \g_@@_width_last_col_dim
3434     \bool_if:NT \l_@@_code_before_bool
3435     {
3436       \pgfsys@markposition
3437       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3438     }
3439     \pgfpicture
3440     \pgfrememberpicturepositiononpagetrue
3441     \pgfcoordinate
3442     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3443     \pgfpointorigin
3444     \str_if_empty:NF \l_@@_name_str
3445     {
3446       \pgfnodealias
3447       {
3448         \l_@@_name_str - col
3449         - \int_eval:n { \g_@@_col_total_int + 1 }
3450       }
3451       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3452     }
3453     \endpgfpicture
3454   }
3455 }
3456 \cr
3457 }

```

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

```

3458 \tl_const:Nn \c_@@_preamble_first_col_tl
3459 {
3460   >
3461   {

```

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).

```

3462   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3463   \bool_gset_true:N \g_@@_after_col_zero_bool
3464   \@@_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.

```

3465   \hbox_set:Nw \l_@@_cell_box
3466   \@@_math_toggle_token:
3467   \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”.

```

3468   \bool_lazy_and:nnT
3469   { \int_compare_p:nNn \c@iRow > 0 }
3470   {
3471     \bool_lazy_or_p:nn
3472     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3473     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3474   }
3475   {
3476     \l_@@_code_for_first_col_tl
3477     \xglobal \colorlet { nicematrix-first-col } { . }
3478   }
3479 }

```

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`.

```

3480   l
3481   <
3482   {
3483     \@@_math_toggle_token:
3484     \hbox_set_end:
3485     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3486     \@@_adjust_size_box:
3487     \@@_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.

```

3488     \dim_gset:Nn \g_@@_width_first_col_dim
3489     { \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.

```

3490     \hbox_overlap_left:n
3491     {
3492       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3493       \@@_node_for_cell:
3494       { \box_use_drop:N \l_@@_cell_box }
3495       \skip_horizontal:N \l_@@_left_delim_dim
3496       \skip_horizontal:N \l_@@_left_margin_dim
3497       \skip_horizontal:N \l_@@_extra_left_margin_dim
3498     }
3499     \bool_gset_false:N \g_@@_empty_cell_bool
3500     \skip_horizontal:N -2\col@sep
3501   }
3502 }

```

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

```

3503 \tl_const:Nn \c_@@_preamble_last_col_tl
3504 {
3505   >
3506   {
3507     \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).

```

3508     \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.

```

3509     \bool_gset_true:N \g_@@_last_col_found_bool
3510     \int_gincr:N \c@jCol
3511     \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.

```

3512     \hbox_set:Nw \l_@@_cell_box
3513     \@@_math_toggle_token:
3514     \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”.

```

3515     \int_compare:nNnT \c@iRow > 0
3516     {
3517       \bool_lazy_or:nnT
3518       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3519       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3520       {
3521         \l_@@_code_for_last_col_tl
3522         \xglobal \colorlet { nicematrix-last-col } { . }
3523       }
3524     }
3525 }

```



```

3526     l
3527     <
3528     {
3529         \@@_math_toggle_token:
3530         \hbox_set_end:
3531         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3532         \@@_adjust_size_box:
3533         \@@_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.

```

3534         \dim_gset:Nn \g_@@_width_last_col_dim
3535         { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3536         \skip_horizontal:N -2\col@sep

```

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

```

3537         \hbox_overlap_right:n
3538         {
3539             \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3540             {
3541                 \skip_horizontal:N \l_@@_right_delim_dim
3542                 \skip_horizontal:N \l_@@_right_margin_dim
3543                 \skip_horizontal:N \l_@@_extra_right_margin_dim
3544                 \@@_node_for_cell:
3545             }
3546         }
3547         \bool_gset_false:N \g_@@_empty_cell_bool
3548     }
3549 }

```

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

```

3550 \NewDocumentEnvironment { NiceArray } { }
3551 {
3552     \bool_gset_false:N \g_@@_delims_bool
3553     \str_if_empty:NT \g_@@_name_env_str
3554     { \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).

```

3555     \NiceArrayWithDelims . .
3556 }
3557 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3558 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3559 {
3560     \NewDocumentEnvironment { #1 NiceArray } { }
3561     {
3562         \bool_gset_true:N \g_@@_delims_bool
3563         \str_if_empty:NT \g_@@_name_env_str
3564         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3565         \@@_test_if_math_mode:
3566         \NiceArrayWithDelims #2 #3
3567     }
3568     { \endNiceArrayWithDelims }
3569 }

3570 \@@_def_env:nnn p ( )
3571 \@@_def_env:nnn b [ ]
3572 \@@_def_env:nnn B \{ \}
3573 \@@_def_env:nnn v | |
3574 \@@_def_env:nnn V \| \|

```

14 The environment {NiceMatrix} and its variants

```

3575 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3576 {
3577   \bool_set_false:N \l_@@_preamble_bool
3578   \tl_clear:N \l_tmpa_tl
3579   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3580     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3581   \tl_put_right:Nn \l_tmpa_tl
3582     {
3583       *
3584       {
3585         \int_case:nnF \l_@@_last_col_int
3586           {
3587             -2 { \c@MaxMatrixCols }
3588             -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).

```

3589       }
3590       { \int_eval:n { \l_@@_last_col_int - 1 } }
3591     }
3592     { #2 }
3593   }
3594   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3595   \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3596 }
3597 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3598 \clist_map_inline:nn { p , b , B , v , V }
3599 {
3600   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3601   {
3602     \bool_gset_true:N \g_@@_delims_bool
3603     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3604     % added 2023/10/01
3605     \int_if_zero:nT \l_@@_last_col_int
3606       {
3607         \bool_set_true:N \l_@@_last_col_without_value_bool
3608         \int_set:Nn \l_@@_last_col_int { -1 }
3609       }
3610     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3611     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3612   }
3613   { \use:c { end #1 NiceArray } }
3614 }

```

We define also an environment {NiceMatrix}

```

3615 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3616 {
3617   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3618   % added 2023/10/01
3619   \int_if_zero:nT \l_@@_last_col_int
3620     {
3621       \bool_set_true:N \l_@@_last_col_without_value_bool
3622       \int_set:Nn \l_@@_last_col_int { -1 }
3623     }
3624   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3625   \bool_lazy_or:nnT
3626     { \clist_if_empty_p:N \l_@@_vlines_clist }
3627     { \l_@@_except_borders_bool }
3628     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3629   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3630 }

```

```
3631 { \endNiceArray }
```

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

```
3632 \cs_new_protected:Npn \l_@@_NotEmpty:
3633 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3634 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3635 {
```

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.

```
3636   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3637   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3638   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3639   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3640   \tl_if_empty:NF \l_@@_short_caption_tl
3641   {
3642     \tl_if_empty:NT \l_@@_caption_tl
3643     {
3644       \@@_error_or_warning:n { short-caption~without~caption }
3645       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3646     }
3647   }
3648   \tl_if_empty:NF \l_@@_label_tl
3649   {
3650     \tl_if_empty:NT \l_@@_caption_tl
3651     { \@@_error_or_warning:n { label~without~caption } }
3652   }
3653   \NewDocumentEnvironment { TabularNote } { b }
3654   {
3655     \bool_if:NTF \l_@@_in_code_after_bool
3656     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3657     {
3658       \tl_if_empty:NF \g_@@_tabularnote_tl
3659       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3660       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3661     }
3662   }
3663   { }
3664   \bool_set_true:N \l_@@_tabular_bool
3665   \NiceArray { #2 }
3666 }
3667 { \endNiceArray }
```

```
3668 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3669 {
3670   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3671   \dim_zero_new:N \l_@@_width_dim
3672   \dim_set:Nn \l_@@_width_dim { #1 }
3673   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3674   \bool_set_true:N \l_@@_tabular_bool
3675   \NiceArray { #3 }
3676 }
3677 {
3678   \endNiceArray
3679   \int_if_zero:nT \g_@@_total_X_weight_int
3680   { \@@_error:n { NiceTabularX~without~X } }
3681 }
```

```

3682 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3683 {
3684   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3685   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3686   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3687   \bool_set_true:N \l_@@_tabular_bool
3688   \NiceArray { #3 }
3689 }
3690 { \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`).

```

3691 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3692 {
3693   \bool_lazy_all:nT
3694   {
3695     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3696     \l_@@_hvlines_bool
3697     { ! \g_@@_delims_bool }
3698     { ! \l_@@_except_borders_bool }
3699   }
3700   {
3701     \bool_set_true:N \l_@@_except_borders_bool
3702     \clist_if_empty:NF \l_@@_corners_clist
3703     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3704     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3705     {
3706       \@@_stroke_block:nnn
3707       {
3708         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3709         draw = \l_@@_rules_color_tl
3710       }
3711       { 1-1 }
3712       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3713     }
3714   }
3715 }

3716 \cs_new_protected:Npn \@@_after_array:
3717 {
3718   \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.

```

3719   \bool_if:NT \g_@@_last_col_found_bool
3720   { \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`.

```

3721   \bool_if:NT \l_@@_last_col_without_value_bool
3722   { \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.

```

3723 \bool_if:NT \l_@@_last_row_without_value_bool
3724 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3725 \tl_build_gput_right:Nx \g_@@_aux_tl
3726 {
3727   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3728   {
3729     \int_use:N \l_@@_first_row_int ,
3730     \int_use:N \c@iRow ,
3731     \int_use:N \g_@@_row_total_int ,
3732     \int_use:N \l_@@_first_col_int ,
3733     \int_use:N \c@jCol ,
3734     \int_use:N \g_@@_col_total_int
3735   }
3736 }

```

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`).

```

3737 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3738 {
3739   \tl_build_gput_right:Nx \g_@@_aux_tl
3740   {
3741     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3742     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3743   }
3744 }
3745 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3746 {
3747   \tl_build_gput_right:Nx \g_@@_aux_tl
3748   {
3749     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3750     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3751     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3752     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3753   }
3754 }

```

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

```

3755 \@@_create_diag_nodes:

```

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

```

3756 \pgfpicture
3757 \int_step_inline:nn \c@iRow
3758 {
3759   \pgfnodealias
3760   { \@@_env: - ##1 - last }
3761   { \@@_env: - ##1 - \int_use:N \c@jCol }
3762 }
3763 \int_step_inline:nn \c@jCol
3764 {
3765   \pgfnodealias
3766   { \@@_env: - last - ##1 }
3767   { \@@_env: - \int_use:N \c@iRow - ##1 }
3768 }
3769 \str_if_empty:NF \l_@@_name_str
3770 {
3771   \int_step_inline:nn \c@iRow
3772   {
3773     \pgfnodealias
3774     { \l_@@_name_str - ##1 - last }
3775     { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

3776     }
3777     \int_step_inline:nn \c@jCol
3778     {
3779         \pgfnodealias
3780         { \l_@@_name_str - last - ##1 }
3781         { \@@_env: - \int_use:N \c@iRow - ##1 }
3782     }
3783 }
3784 \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.

```

3785     \bool_if:NT \l_@@_parallelize_diags_bool
3786     {
3787         \int_gzero_new:N \g_@@_ddots_int
3788         \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.

```

3789         \dim_gzero_new:N \g_@@_delta_x_one_dim
3790         \dim_gzero_new:N \g_@@_delta_y_one_dim
3791         \dim_gzero_new:N \g_@@_delta_x_two_dim
3792         \dim_gzero_new:N \g_@@_delta_y_two_dim
3793     }
3794     \int_zero_new:N \l_@@_initial_i_int
3795     \int_zero_new:N \l_@@_initial_j_int
3796     \int_zero_new:N \l_@@_final_i_int
3797     \int_zero_new:N \l_@@_final_j_int
3798     \bool_set_false:N \l_@@_initial_open_bool
3799     \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.

```

3800     \bool_if:NT \l_@@_small_bool
3801     {
3802         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3803         \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.

```

3804         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3805         { 0.6 \l_@@_xdots_shorten_start_dim }
3806         \dim_set:Nn \l_@@_xdots_shorten_end_dim
3807         { 0.6 \l_@@_xdots_shorten_end_dim }
3808     }

```

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

```

3809     \@@_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.

```

3810     \@@_compute_corners:

```

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

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

```

3811 \@@_adjust_pos_of_blocks_seq:
3812 \@@_deal_with_rounded_corners:
3813 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3814 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3815 \IfPackageLoadedTF { tikz }
3816 {
3817   \tikzset
3818   {
3819     every~picture / .style =
3820     {
3821       overlay ,
3822       remember~picture ,
3823       name~prefix = \@@_env: -
3824     }
3825   }
3826 }
3827 { }
3828 \cs_set_eq:NN \ialign \@@_old_ialign:
3829 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3830 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3831 \cs_set_eq:NN \OverBrace \@@_OverBrace
3832 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3833 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3834 \cs_set_eq:NN \line \@@_line
3835 \g_@@_pre_code_after_tl
3836 \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.

```

3837 \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).

```

3838 \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`).

```

3839 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3840 { \@@_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:`.

```

3841 \bool_set_true:N \l_@@_in_code_after_bool
3842 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3843 \scan_stop:
3844 \tl_gclear:N \g_nicematrix_code_after_tl
3845 \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.

```

3846 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3847 \tl_if_empty:NF \g_@@_pre_code_before_tl
3848 {
3849   \tl_build_gput_right:Nx \g_@@_aux_tl

```

```

3850     {
3851         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3852         { \exp_not:V \g_@@_pre_code_before_tl }
3853     }
3854     \tl_gclear:N \g_@@_pre_code_before_tl
3855 }
3856 \tl_if_empty:NF \g_nicematrix_code_before_tl
3857 {
3858     \tl_build_gput_right:Nx \g_@@_aux_tl
3859     {
3860         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3861         { \exp_not:V \g_nicematrix_code_before_tl }
3862     }
3863     \tl_gclear:N \g_nicematrix_code_before_tl
3864 }

3865 \str_gclear:N \g_@@_name_env_str
3866 \@@_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`.

```

3867     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3868 }

```

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`.

```

3869 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3870 { \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).

```

3871 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3872 {
3873     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3874     { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3875 }

```

The following command must *not* be protected.

```

3876 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3877 {
3878     { #1 }
3879     { #2 }
3880     {
3881         \int_compare:nNnTF { #3 } > { 99 }
3882         { \int_use:N \c@iRow }
3883         { #3 }
3884     }
3885     {
3886         \int_compare:nNnTF { #4 } > { 99 }
3887         { \int_use:N \c@jCol }
3888         { #4 }

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`


```

3889     }
3890     { #5 }
3891 }

```

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).

```

3892 \hook_gput_code:nnn { begindocument } { . }
3893 {
3894   \cs_new_protected:Npx \@@_draw_dotted_lines:
3895   {
3896     \c_@@_pgfortikzpicture_tl
3897     \@@_draw_dotted_lines_i:
3898     \c_@@_endpgfortikzpicture_tl
3899   }
3900 }

```

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

```

3901 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3902 {
3903   \pgfrememberpicturepositiononpagetrue
3904   \pgf@relevantforpicturesizefalse
3905   \g_@@_HVdotsfor_lines_tl
3906   \g_@@_Vdots_lines_tl
3907   \g_@@_Ddots_lines_tl
3908   \g_@@_Iddots_lines_tl
3909   \g_@@_Cdots_lines_tl
3910   \g_@@_Ldots_lines_tl
3911 }
3912 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3913 {
3914   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3915   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3916 }

```

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

```

3917 \pgfdeclareshape { @@_diag_node }
3918 {
3919   \savedanchor { \five }
3920   {
3921     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3922     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3923   }
3924   \anchor { 5 } { \five }
3925   \anchor { center } { \pgfpointorigin }
3926 }

```

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).

```

3927 \cs_new_protected:Npn \@@_create_diag_nodes:
3928 {
3929   \pgfpicture
3930   \pgfrememberpicturepositiononpagetrue
3931   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3932   {
3933     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3934     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3935     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }

```

```

3936 \dim_set_eq:NN \l_tmpb_dim \pgf@y
3937 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3938 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3939 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3940 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3941 \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.

```

3942 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3943 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3944 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3945 \str_if_empty:NF \l_@@_name_str
3946 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3947 }

```

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

```

3948 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3949 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3950 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3951 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3952 \pgfcoordinate
3953 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3954 \pgfnodealias
3955 { \@@_env: - last }
3956 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3957 \str_if_empty:NF \l_@@_name_str
3958 {
3959 \pgfnodealias
3960 { \l_@@_name_str - \int_use:N \l_tmpa_int }
3961 { \@@_env: - \int_use:N \l_tmpa_int }
3962 \pgfnodealias
3963 { \l_@@_name_str - last }
3964 { \@@_env: - last }
3965 }
3966 \endpgfpicture
3967 }

```

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 `\@@_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.

```

3968 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3969 {

```

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

```

3970 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

3971 \int_set:Nn \l_@@_initial_i_int { #1 }
3972 \int_set:Nn \l_@@_initial_j_int { #2 }
3973 \int_set:Nn \l_@@_final_i_int { #1 }
3974 \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.

```

3975 \bool_set_false:N \l_@@_stop_loop_bool
3976 \bool_do_until:Nn \l_@@_stop_loop_bool
3977 {
3978   \int_add:Nn \l_@@_final_i_int { #3 }
3979   \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3980   \bool_set_false:N \l_@@_final_open_bool
3981   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3982   {
3983     \int_compare:nNnTF { #3 } = 1
3984     { \bool_set_true:N \l_@@_final_open_bool }
3985     {
3986       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3987       { \bool_set_true:N \l_@@_final_open_bool }
3988     }
3989   }
3990   {
3991     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3992     {
3993       \int_compare:nNnT { #4 } = { -1 }
3994       { \bool_set_true:N \l_@@_final_open_bool }
3995     }
3996     {
3997       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3998       {
3999         \int_compare:nNnT { #4 } = 1
4000         { \bool_set_true:N \l_@@_final_open_bool }
4001       }
4002     }
4003   }
4004   \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.

```

4005   {

```

We do a step backwards.

```

4006     \int_sub:Nn \l_@@_final_i_int { #3 }
4007     \int_sub:Nn \l_@@_final_j_int { #4 }
4008     \bool_set_true:N \l_@@_stop_loop_bool
4009   }

```

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`.

```

4010     {
4011         \cs_if_exist:cTF
4012         {
4013             @@ _ dotted _
4014             \int_use:N \l_@@_final_i_int -
4015             \int_use:N \l_@@_final_j_int
4016         }
4017         {
4018             \int_sub:Nn \l_@@_final_i_int { #3 }
4019             \int_sub:Nn \l_@@_final_j_int { #4 }
4020             \bool_set_true:N \l_@@_final_open_bool
4021             \bool_set_true:N \l_@@_stop_loop_bool
4022         }
4023     }
4024     \cs_if_exist:cTF
4025     {
4026         pgf @ sh @ ns @ \@@_env:
4027         - \int_use:N \l_@@_final_i_int
4028         - \int_use:N \l_@@_final_j_int
4029     }
4030     { \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.

```

4031     {
4032         \cs_set:cpn
4033         {
4034             @@ _ dotted _
4035             \int_use:N \l_@@_final_i_int -
4036             \int_use:N \l_@@_final_j_int
4037         }
4038         { }
4039     }
4040 }
4041 }
4042 }

```

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

```

4043 \bool_set_false:N \l_@@_stop_loop_bool
4044 \bool_do_until:Nn \l_@@_stop_loop_bool
4045 {
4046     \int_sub:Nn \l_@@_initial_i_int { #3 }
4047     \int_sub:Nn \l_@@_initial_j_int { #4 }
4048     \bool_set_false:N \l_@@_initial_open_bool
4049     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4050     {
4051         \int_compare:nNnTF { #3 } = 1
4052         { \bool_set_true:N \l_@@_initial_open_bool }
4053         {
4054             \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4055             { \bool_set_true:N \l_@@_initial_open_bool }
4056         }
4057     }
4058     {
4059         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4060         {

```

```

4061         \int_compare:nNt { #4 } = 1
4062         { \bool_set_true:N \l_@@_initial_open_bool }
4063     }
4064     {
4065         \int_compare:nNt \l_@@_initial_j_int > \l_@@_col_max_int
4066         {
4067             \int_compare:nNt { #4 } = { -1 }
4068             { \bool_set_true:N \l_@@_initial_open_bool }
4069         }
4070     }
4071 }
4072 \bool_if:NTF \l_@@_initial_open_bool
4073 {
4074     \int_add:Nn \l_@@_initial_i_int { #3 }
4075     \int_add:Nn \l_@@_initial_j_int { #4 }
4076     \bool_set_true:N \l_@@_stop_loop_bool
4077 }
4078 {
4079     \cs_if_exist:cTF
4080     {
4081         @@ _ dotted _
4082         \int_use:N \l_@@_initial_i_int -
4083         \int_use:N \l_@@_initial_j_int
4084     }
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_@@_initial_open_bool
4089         \bool_set_true:N \l_@@_stop_loop_bool
4090     }
4091     {
4092         \cs_if_exist:cTF
4093         {
4094             pgf @ sh @ ns @ \@@_env:
4095             - \int_use:N \l_@@_initial_i_int
4096             - \int_use:N \l_@@_initial_j_int
4097         }
4098         { \bool_set_true:N \l_@@_stop_loop_bool }
4099     }
4100     \cs_set:cpn
4101     {
4102         @@ _ dotted _
4103         \int_use:N \l_@@_initial_i_int -
4104         \int_use:N \l_@@_initial_j_int
4105     }
4106     { }
4107 }
4108 }
4109 }
4110 }

```

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.

```

4111 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4112 {
4113     { \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`.

```

4114     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4115     { \int_use:N \l_@@_final_i_int }
4116     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4117     { } % for the name of the block
4118 }

```

```
4119 }
```

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.

```
4120 \cs_new_protected:Npn \@@_open_shorten:
4121 {
4122   \bool_if:NT \l_@@_initial_open_bool
4123   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4124   \bool_if:NT \l_@@_final_open_bool
4125   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4126 }
```

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).

```
4127 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4128 {
4129   \int_set:Nn \l_@@_row_min_int 1
4130   \int_set:Nn \l_@@_col_min_int 1
4131   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4132   \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`.

```
4133 \seq_map_inline:Nn \g_@@_submatrix_seq
4134 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4135 }
```

`#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.

```
4136 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4137 {
4138   \bool_if:nT
4139   {
4140     \int_compare_p:n { #3 <= #1 }
4141     && \int_compare_p:n { #1 <= #5 }
4142     && \int_compare_p:n { #4 <= #2 }
4143     && \int_compare_p:n { #2 <= #6 }
4144   }
4145   {
4146     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4147     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4148     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4149     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4150   }
4151 }
```

```
4152 \cs_new_protected:Npn \@@_set_initial_coords:
4153 {
4154   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4155   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4156 }
4157 \cs_new_protected:Npn \@@_set_final_coords:
4158 {
4159   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4160   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
```

```

4161 }
4162 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4163 {
4164   \pgfpointanchor
4165   {
4166     \@@_env:
4167     - \int_use:N \l_@@_initial_i_int
4168     - \int_use:N \l_@@_initial_j_int
4169   }
4170   { #1 }
4171   \@@_set_initial_coords:
4172 }
4173 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4174 {
4175   \pgfpointanchor
4176   {
4177     \@@_env:
4178     - \int_use:N \l_@@_final_i_int
4179     - \int_use:N \l_@@_final_j_int
4180   }
4181   { #1 }
4182   \@@_set_final_coords:
4183 }
4184 \cs_new_protected:Npn \@@_open_x_initial_dim:
4185 {
4186   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4187   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4188   {
4189     \cs_if_exist:cT
4190     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4191     {
4192       \pgfpointanchor
4193       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4194       { west }
4195       \dim_set:Nn \l_@@_x_initial_dim
4196       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4197     }
4198   }

```

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

```

4199   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4200   {
4201     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4202     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4203     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4204   }
4205 }
4206 \cs_new_protected:Npn \@@_open_x_final_dim:
4207 {
4208   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4209   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4210   {
4211     \cs_if_exist:cT
4212     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4213     {
4214       \pgfpointanchor
4215       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4216       { east }
4217       \dim_set:Nn \l_@@_x_final_dim
4218       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4219     }
4220   }

```

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

```

4221 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4222 {
4223   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4224   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4225   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4226 }
4227 }
```

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.

```

4228 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4229 {
4230   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4231   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4232   {
4233     \@@_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.

```

4234     \group_begin:
4235     \@@_open_shorten:
4236     \int_if_zero:nTF { #1 }
4237     { \color { nicematrix-first-row } }
4238     {
```

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.

```

4239         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4240         { \color { nicematrix-last-row } }
4241     }
4242     \keys_set:nn { NiceMatrix / xdots } { #3 }
4243     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4244     \@@_actually_draw_Ldots:
4245   \group_end:
4246 }
4247 }
```

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`.

```

4248 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4249 {
4250   \bool_if:NTF \l_@@_initial_open_bool
4251   {
4252     \@@_open_x_initial_dim:
4253     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4254     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4255   }
4256   { \@@_set_initial_coords_from_anchor:n { base-east } }
4257   \bool_if:NTF \l_@@_final_open_bool
4258   {
```



```

4259 \@@_open_x_final_dim:
4260 \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4261 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4262 }
4263 { \@@_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.

```

4264 \bool_lazy_all:nTF
4265 {
4266   \l_@@_initial_open_bool
4267   \l_@@_final_open_bool
4268   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4269 }
4270 {
4271   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4272   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4273 }

```

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 (?).

```

4274 {
4275   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4276   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4277 }
4278 \@@_draw_line:
4279 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4280 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4281 {
4282   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4283   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4284   {
4285     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 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.

```

4286 \group_begin:
4287 \@@_open_shorten:
4288 \int_if_zero:nTF { #1 }
4289 { \color { nicematrix-first-row } }
4290 {

```

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.

```

4291 \int_compare:nNnT { #1 } = \l_@@_last_row_int
4292 { \color { nicematrix-last-row } }
4293 }
4294 \keys_set:nn { NiceMatrix / xdots } { #3 }
4295 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4296 \@@_actually_draw_Cdots:
4297 \group_end:
4298 }
4299 }

```

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.

```

4300 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4301 {
4302   \bool_if:NTF \l_@@_initial_open_bool
4303     { \@@_open_x_initial_dim: }
4304     { \@@_set_initial_coords_from_anchor:n { mid~east } }
4305   \bool_if:NTF \l_@@_final_open_bool
4306     { \@@_open_x_final_dim: }
4307     { \@@_set_final_coords_from_anchor:n { mid~west } }
4308   \bool_lazy_and:nnTF
4309     \l_@@_initial_open_bool
4310     \l_@@_final_open_bool
4311   {
4312     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4313     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4314     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4315     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4316     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4317   }
4318   {
4319     \bool_if:NT \l_@@_initial_open_bool
4320       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4321     \bool_if:NT \l_@@_final_open_bool
4322       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4323   }
4324   \@@_draw_line:
4325 }

4326 \cs_new_protected:Npn \@@_open_y_initial_dim:
4327 {
4328   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4329   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4330   {
4331     \cs_if_exist:cT
4332       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4333     {
4334       \pgfpointanchor
4335         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4336         { north }
4337       \dim_set:Nn \l_@@_y_initial_dim
4338         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4339     }
4340   }
4341   % modified 2023-08-10
4342   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4343   {
4344     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4345     \dim_set:Nn \l_@@_y_initial_dim
4346     {
4347       \fp_to_dim:n
4348       {
4349         \pgf@y
4350         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4351       }
4352     }
4353   }
4354 }

4355 \cs_new_protected:Npn \@@_open_y_final_dim:

```

```

4356 {
4357   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4358   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4359   {
4360     \cs_if_exist:cT
4361     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4362     {
4363       \pgfpointanchor
4364       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4365       { south }
4366       \dim_set:Nn \l_@@_y_final_dim
4367       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4368     }
4369   }
4370   % modified 2023-08-10
4371   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4372   {
4373     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4374     \dim_set:Nn \l_@@_y_final_dim
4375     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4376   }
4377 }

```

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.

```

4378 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4379 {
4380   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4381   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4382   {
4383     \@@_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.

```

4384   \group_begin:
4385   \@@_open_shorten:
4386   \int_if_zero:nTF { #2 }
4387   { \color { nicematrix-first-col } }
4388   {
4389     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4390     { \color { nicematrix-last-col } }
4391   }
4392   \keys_set:nn { NiceMatrix / xdots } { #3 }
4393   \tl_if_empty:VF \l_@@_xdots_color_tl
4394   { \color { \l_@@_xdots_color_tl } }
4395   \@@_actually_draw_Vdots:
4396   \group_end:
4397 }
4398 }

```

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`.

```
4399 \cs_new_protected:Npn \l_@@_actually_draw_Vdots:
4400 {
```

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

```
4401 \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.

```
4402 {
4403   \@@_open_y_initial_dim:
4404   \@@_open_y_final_dim:
4405   \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4406 {
4407   \@@_qpoint:n { col - 1 }
4408   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4409   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4410   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4411   % \bool_if:NT \g_@@_delims_bool
4412   % {
4413     \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4414   % }
4415 }
4416 {
4417   \bool_lazy_and:nnTF
4418   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4419   { \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”.

```
4420 {
4421   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4422   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4423   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4424   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4425   % \bool_if:NT \g_@@_delims_bool
4426   % {
4427     \dim_add:Nn
4428     \l_@@_x_initial_dim
4429     \c_@@_shift_exterior_Vdots_dim
4430   % }
4431 }
```

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

```
4432 {
4433   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4434   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4435   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4436   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4437 }
4438 }
4439 }
```

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.

```
4440 {
4441   \bool_set_false:N \l_tmpa_bool
4442   \bool_lazy_and:nnT
4443   { ! \l_@@_initial_open_bool }
4444   { ! \l_@@_final_open_bool }
4445   {
4446     \@@_set_initial_coords_from_anchor:n { south-west }
4447     \@@_set_final_coords_from_anchor:n { north-west }
4448     \bool_set:Nn \l_tmpa_bool
4449     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4450   }
```

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

```

4451 \bool_if:NTF \l_@@_initial_open_bool
4452 {
4453   \@@_open_y_initial_dim:
4454   \@@_set_final_coords_from_anchor:n { north }
4455   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4456 }
4457 {
4458   \@@_set_initial_coords_from_anchor:n { south }
4459   \bool_if:NTF \l_@@_final_open_bool
4460   \@@_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.

```

4461 {
4462   \@@_set_final_coords_from_anchor:n { north }
4463   \dim_compare:nNf \l_@@_x_initial_dim = \l_@@_x_final_dim
4464   {
4465     \dim_set:Nn \l_@@_x_initial_dim
4466     {
4467       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4468       \l_@@_x_initial_dim \l_@@_x_final_dim
4469     }
4470   }
4471 }
4472 }
4473 }
4474 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4475 \@@_draw_line:
4476 }

```

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.

```

4477 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4478 {
4479   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4480   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4481   {
4482     \@@_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.

```

4483 \group_begin:
4484 \@@_open_shorten:
4485 \keys_set:nn { NiceMatrix / xdots } { #3 }
4486 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4487 \@@_actually_draw_Ddots:
4488 \group_end:
4489 }
4490 }

```

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`.

```

4491 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4492 {
4493   \bool_if:NTF \l_@@_initial_open_bool
4494   {
4495     \@@_open_y_initial_dim:
4496     \@@_open_x_initial_dim:
4497   }
4498   { \@@_set_initial_coords_from_anchor:n { south-east } }
4499   \bool_if:NTF \l_@@_final_open_bool
4500   {
4501     \@@_open_x_final_dim:
4502     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4503   }
4504   { \@@_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.

```

4505   \bool_if:NT \l_@@_parallelize_diags_bool
4506   {
4507     \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).

```

4508     \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.

```

4509     {
4510       \dim_gset:Nn \g_@@_delta_x_one_dim
4511       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4512       \dim_gset:Nn \g_@@_delta_y_one_dim
4513       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4514     }

```

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`.

```

4515     {
4516       \dim_set:Nn \l_@@_y_final_dim
4517       {
4518         \l_@@_y_initial_dim +
4519         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4520         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4521       }
4522     }
4523   }
4524   \@@_draw_line:
4525 }

```

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.

```

4526 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4527 {
4528   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4529   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4530   {
4531     \@@_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.

```

4532     \group_begin:
4533     \@@_open_shorten:
4534     \keys_set:nn { NiceMatrix / xdots } { #3 }
4535     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4536     \@@_actually_draw_Iddots:
4537     \group_end:
4538   }
4539 }

```

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.`

```

4540 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4541 {
4542   \bool_if:NTF \l_@@_initial_open_bool
4543   {
4544     \@@_open_y_initial_dim:
4545     \@@_open_x_initial_dim:
4546   }
4547   { \@@_set_initial_coords_from_anchor:n { south-west } }
4548   \bool_if:NTF \l_@@_final_open_bool
4549   {
4550     \@@_open_y_final_dim:
4551     \@@_open_x_final_dim:
4552   }
4553   { \@@_set_final_coords_from_anchor:n { north-east } }
4554   \bool_if:NT \l_@@_parallelize_diags_bool
4555   {
4556     \int_gincr:N \g_@@_iddots_int
4557     \int_compare:nNnTF \g_@@_iddots_int = 1
4558     {
4559       \dim_gset:Nn \g_@@_delta_x_two_dim
4560       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4561       \dim_gset:Nn \g_@@_delta_y_two_dim
4562       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4563     }
4564     {
4565       \dim_set:Nn \l_@@_y_final_dim
4566       {
4567         \l_@@_y_initial_dim +
4568         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4569         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4570       }
4571     }
4572   }
4573   \@@_draw_line:
4574 }

```

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`

```

4575 \cs_new_protected:Npn \@@_draw_line:
4576 {
4577   \pgfrememberpicturepositiononpagetrue
4578   \pgf@relevantforpicturesizefalse
4579   \bool_lazy_or:nnTF
4580     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4581     \l_@@_dotted_bool
4582     \@@_draw_standard_dotted_line:
4583     \@@_draw_unstandard_dotted_line:
4584 }

```

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.

```

4585 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4586 {
4587   \begin { scope }
4588   \@@_draw_unstandard_dotted_line:o
4589   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4590 }

```

We have used the fact that, in PGF, a 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.

```

4591 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4592 {
4593   \@@_draw_unstandard_dotted_line:nVWV
4594   { #1 }
4595   \l_@@_xdots_up_tl
4596   \l_@@_xdots_down_tl
4597   \l_@@_xdots_middle_tl
4598 }
4599 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4600 \hook_gput_code:nnn { begindocument } { . }
4601 {
4602   \IfPackageLoadedTF { tikz }
4603   {
4604     \tikzset
4605     {
4606       @@_node_above / .style = { sloped , above } ,
4607       @@_node_below / .style = { sloped , below } ,
4608       @@_node_middle / .style =
4609       {

```



```

4610         sloped ,
4611         inner~sep = \c_@@_innersep_middle_dim
4612     }
4613 }
4614 }
4615 { }
4616 }

```

```

4617 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4618 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shorten >` and `shorten <` of TikZ in the command `\draw`, we don’t have the expected output with `{decorate,decoration=brace}` is used.

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4619     \dim_zero_new:N \l_@@_l_dim
4620     \dim_set:Nn \l_@@_l_dim
4621     {
4622         \fp_to_dim:n
4623         {
4624             sqrt
4625             (
4626                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4627                 +
4628                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4629             )
4630         }
4631     }
4632     \bool_lazy_and:nnT % security
4633     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } < \c_@@_max_l_dim }
4634     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > { 1 pt } }
4635     {
4636         \dim_set:Nn \l_tmpa_dim
4637         {
4638             \l_@@_x_initial_dim
4639             + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4640             * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4641         }
4642         \dim_set:Nn \l_tmpb_dim
4643         {
4644             \l_@@_y_initial_dim
4645             + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4646             * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4647         }
4648         \dim_set:Nn \l_@@_tmpc_dim
4649         {
4650             \l_@@_x_final_dim
4651             - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4652             * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4653         }
4654         \dim_set:Nn \l_@@_tmpd_dim
4655         {
4656             \l_@@_y_final_dim
4657             - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4658             * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4659         }
4660         \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4661         \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4662         \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4663         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4664     }

```

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

```

4665 \bool_if:NT \l_@@_xdots_h_labels_bool
4666 {
4667   \tikzset
4668   {
4669     @@_node_above / .style = { auto = left } ,
4670     @@_node_below / .style = { auto = right } ,
4671     @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4672   }
4673 }
4674 \tl_if_empty:nF { #4 }
4675 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4676 \draw
4677 [ #1 ]
4678 ( \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).

```

4679 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4680 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4681 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4682 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4683 \end { scope }
4684 }
4685 \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).

```

4686 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4687 {
4688   \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.

```

4689 \dim_zero_new:N \l_@@_l_dim
4690 \dim_set:Nn \l_@@_l_dim
4691 {
4692   \fp_to_dim:n
4693   {
4694     sqrt
4695     (
4696       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4697       +
4698       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4699     )
4700   }
4701 }

```

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.

```

4702 \bool_lazy_or:nnF
4703 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4704 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4705 \@@_draw_standard_dotted_line_i:
4706 \group_end:
4707 \bool_lazy_all:nF
4708 {
4709   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4710   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4711   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4712 }

```

```

4713 \l_@@_labels_standard_dotted_line:
4714 }
4715 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4716 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4717 {

```

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

```

4718 \int_set:Nn \l_tmpa_int
4719 {
4720 \dim_ratio:nn
4721 {
4722 \l_@@_l_dim
4723 - \l_@@_xdots_shorten_start_dim
4724 - \l_@@_xdots_shorten_end_dim
4725 }
4726 \l_@@_xdots_inter_dim
4727 }

```

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

```

4728 \dim_set:Nn \l_tmpa_dim
4729 {
4730 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4731 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4732 }
4733 \dim_set:Nn \l_tmpb_dim
4734 {
4735 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4736 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4737 }

```

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.

```

4738 \dim_gadd:Nn \l_@@_x_initial_dim
4739 {
4740 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4741 \dim_ratio:nn
4742 {
4743 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4744 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4745 }
4746 { 2 \l_@@_l_dim }
4747 }
4748 \dim_gadd:Nn \l_@@_y_initial_dim
4749 {
4750 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4751 \dim_ratio:nn
4752 {
4753 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4754 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4755 }
4756 { 2 \l_@@_l_dim }
4757 }
4758 \pgf@relevantforpicturesizefalse
4759 \int_step_inline:nnn 0 \l_tmpa_int
4760 {
4761 \pgfpathcircle
4762 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4763 { \l_@@_xdots_radius_dim }
4764 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4765 \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4766 }
4767 \pgfusepathqfill
4768 }

```

```

4769 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4770 {
4771   \pgfscope
4772   \pgftransformshift
4773   {
4774     \pgfpointlineattime { 0.5 }
4775     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4776     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4777   }
4778   \fp_set:Nn \l_tmpa_fp
4779   {
4780     atand
4781     (
4782       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4783       \l_@@_x_final_dim - \l_@@_x_initial_dim
4784     )
4785   }
4786   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4787   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4788   \tl_if_empty:NF \l_@@_xdots_middle_tl
4789   {
4790     \begin { pgfscope }
4791     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4792     \pgfnode
4793     { rectangle }
4794     { center }
4795     {
4796       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4797       {
4798         \c_math_toggle_token
4799         \scriptstyle \l_@@_xdots_middle_tl
4800         \c_math_toggle_token
4801       }
4802     }
4803     { }
4804     {
4805       \pgfsetfillcolor { white }
4806       \pgfusepath { fill }
4807     }
4808     \end { pgfscope }
4809   }
4810   \tl_if_empty:NF \l_@@_xdots_up_tl
4811   {
4812     \pgfnode
4813     { rectangle }
4814     { south }
4815     {
4816       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4817       {
4818         \c_math_toggle_token
4819         \scriptstyle \l_@@_xdots_up_tl
4820         \c_math_toggle_token
4821       }
4822     }
4823     { }
4824     { \pgfusepath { } }
4825   }
4826   \tl_if_empty:NF \l_@@_xdots_down_tl
4827   {
4828     \pgfnode
4829     { rectangle }
4830     { north }
4831     {

```

```

4832         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4833         {
4834             \c_math_toggle_token
4835             \scriptstyle \l_@@_xdots_down_tl
4836             \c_math_toggle_token
4837         }
4838     }
4839     { }
4840     { \pgfusepath { } }
4841 }
4842 \endpgfscope
4843 }

```

19 User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4844 \hook_gput_code:nnn { begindocument } { . }
4845 {
4846     \tl_set:Nn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4847     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4848     \cs_new_protected:Npn \@@_Ldots
4849     { \@@_collect_options:n { \@@_Ldots_i } }
4850     \exp_args:NNV \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4851     {
4852         \int_if_zero:nTF \c@jCol
4853         { \@@_error:nn { in~first~col } \Ldots }
4854         {
4855             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4856             { \@@_error:nn { in~last~col } \Ldots }
4857             {
4858                 \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
4859                 { #1 , down = #2 , up = #3 , middle = #4 }
4860             }
4861         }
4862         \bool_if:NF \l_@@_nullify_dots_bool
4863         { \phantom { \ensuremath { \@@_old_ldots } } }
4864         \bool_gset_true:N \g_@@_empty_cell_bool
4865     }

4866     \cs_new_protected:Npn \@@_Cdots
4867     { \@@_collect_options:n { \@@_Cdots_i } }
4868     \exp_args:NNV \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4869     {
4870         \int_if_zero:nTF \c@jCol
4871         { \@@_error:nn { in~first~col } \Cdots }
4872         {
4873             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4874             { \@@_error:nn { in~last~col } \Cdots }
4875             {
4876                 \@@_instruction_of_type:nnn \c_false_bool { \Cdots }

```

```

4877         { #1 , down = #2 , up = #3 , middle = #4 }
4878     }
4879 }
4880 \bool_if:NF \l_@@_nullify_dots_bool
4881 { \phantom { \ensuremath { \@@_old_cdots } } }
4882 \bool_gset_true:N \g_@@_empty_cell_bool
4883 }

4884 \cs_new_protected:Npn \@@_Vdots
4885 { \@@_collect_options:n { \@@_Vdots_i } }
4886 \exp_args:NNV \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4887 {
4888     \int_if_zero:nTF \c@iRow
4889     { \@@_error:nn { in~first~row } \Vdots }
4890     {
4891         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4892         { \@@_error:nn { in~last~row } \Vdots }
4893         {
4894             \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4895             { #1 , down = #2 , up = #3 , middle = #4 }
4896         }
4897     }
4898     \bool_if:NF \l_@@_nullify_dots_bool
4899     { \phantom { \ensuremath { \@@_old_vdots } } }
4900     \bool_gset_true:N \g_@@_empty_cell_bool
4901 }

4902 \cs_new_protected:Npn \@@_Ddots
4903 { \@@_collect_options:n { \@@_Ddots_i } }
4904 \exp_args:NNV \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4905 {
4906     \int_case:nnF \c@iRow
4907     {
4908         0 { \@@_error:nn { in~first~row } \Ddots }
4909         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4910     }
4911     {
4912         \int_case:nnF \c@jCol
4913         {
4914             0 { \@@_error:nn { in~first~col } \Ddots }
4915             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4916         }
4917         {
4918             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4919             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
4920             { #1 , down = #2 , up = #3 , middle = #4 }
4921         }
4922     }
4923 }
4924 \bool_if:NF \l_@@_nullify_dots_bool
4925 { \phantom { \ensuremath { \@@_old_ddots } } }
4926 \bool_gset_true:N \g_@@_empty_cell_bool
4927 }

4928 \cs_new_protected:Npn \@@_Iddots
4929 { \@@_collect_options:n { \@@_Iddots_i } }
4930 \exp_args:NNV \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4931 {
4932     \int_case:nnF \c@iRow
4933     {
4934         0 { \@@_error:nn { in~first~row } \Iddots }

```

```

4935 \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4936 }
4937 {
4938 \int_case:nnF \c@jCol
4939 {
4940 0 { \@@_error:nn { in~first~col } \Iddots }
4941 \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4942 }
4943 {
4944 \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4945 \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4946 { #1 , down = #2 , up = #3 , middle = #4 }
4947 }
4948 }
4949 \bool_if:NF \l_@@_nullify_dots_bool
4950 { \phantom { \ensuremath { \@@_old_iddots } } }
4951 \bool_gset_true:N \g_@@_empty_cell_bool
4952 }
4953 }

```

End of the `\AddToHook`.

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

```

4954 \keys_define:nn { NiceMatrix / Ddots }
4955 {
4956 draw-first .bool_set:N = \l_@@_draw_first_bool ,
4957 draw-first .default:n = true ,
4958 draw-first .value_forbidden:n = true
4959 }

```

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

```

4960 \cs_new_protected:Npn \@@_Hspace:
4961 {
4962 \bool_gset_true:N \g_@@_empty_cell_bool
4963 \hspace
4964 }

```

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`.

```

4965 \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`.

```

4966 \cs_new:Npn \@@_Hdotsfor:
4967 {
4968 \bool_lazy_and:nnTF
4969 { \int_if_zero_p:n \c@jCol }
4970 { \int_if_zero_p:n \l_@@_first_col_int }
4971 {
4972 \bool_if:NTF \g_@@_after_col_zero_bool
4973 {
4974 \multicolumn { 1 } { c } { }
4975 \@@_Hdotsfor_i
4976 }
4977 { \@@_fatal:n { Hdotsfor~in~col~0 } }
4978 }
4979 {
4980 \multicolumn { 1 } { c } { }
4981 \@@_Hdotsfor_i
4982 }
4983 }

```

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:`).

```

4984 \hook_gput_code:nnn { begindocument } { . }
4985 {
4986   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4987   \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.

```

4988   \cs_new_protected:Npn \@@_Hdotsfor_i
4989     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
4990   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
4991     {
4992       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4993       {
4994         \@@_Hdotsfor:nnnn
4995         { \int_use:N \c@iRow }
4996         { \int_use:N \c@jCol }
4997         { #2 }
4998         {
4999           #1 , #3 ,
5000           down = \exp_not:n { #4 } ,
5001           up = \exp_not:n { #5 } ,
5002           middle = \exp_not:n { #6 }
5003         }
5004       }
5005       \prg_replicate:nn { #2 - 1 }
5006       {
5007         &
5008         \multicolumn { 1 } { c } { }
5009         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i: % added 2023-08-26
5010       }
5011     }
5012 }

```

```

5013 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5014 {
5015   \bool_set_false:N \l_@@_initial_open_bool
5016   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5017   \int_set:Nn \l_@@_initial_i_int { #1 }
5018   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5019   \int_compare:nNnTF { #2 } = 1
5020   {
5021     \int_set:Nn \l_@@_initial_j_int 1
5022     \bool_set_true:N \l_@@_initial_open_bool
5023   }
5024   {
5025     \cs_if_exist:cTF
5026     {
5027       pgf @ sh @ ns @ \@@_env:
5028       - \int_use:N \l_@@_initial_i_int
5029       - \int_eval:n { #2 - 1 }
5030     }
5031     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5032     {
5033       \int_set:Nn \l_@@_initial_j_int { #2 }
5034       \bool_set_true:N \l_@@_initial_open_bool
5035     }
5036   }

```



```

5037 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5038 {
5039   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5040   \bool_set_true:N \l_@@_final_open_bool
5041 }
5042 {
5043   \cs_if_exist:cTF
5044   {
5045     pgf @ sh @ ns @ \@@_env:
5046     - \int_use:N \l_@@_final_i_int
5047     - \int_eval:n { #2 + #3 }
5048   }
5049   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5050   {
5051     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5052     \bool_set_true:N \l_@@_final_open_bool
5053   }
5054 }
5055 \group_begin:
5056 \@@_open_shorten:
5057 \int_if_zero:nTF { #1 }
5058 { \color { nicematrix-first-row } }
5059 {
5060   \int_compare:nNnT { #1 } = \g_@@_row_total_int
5061   { \color { nicematrix-last-row } }
5062 }
5063
5064 \keys_set:nn { NiceMatrix / xdots } { #4 }
5065 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5066 \@@_actually_draw_Ldots:
5067 \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).

```

5068 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5069 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5070 }

5071 \hook_gput_code:nnn { begindocument } { . }
5072 {
5073   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5074   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5075   \cs_new_protected:Npn \@@_Vdotsfor:
5076   { \@@_collect_options:n { \@@_Vdotsfor_i } }
5077   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5078   {
5079     \bool_gset_true:N \g_@@_empty_cell_bool
5080     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5081     {
5082       \@@_Vdotsfor:nnnn
5083       { \int_use:N \c@iRow }
5084       { \int_use:N \c@jCol }
5085       { #2 }
5086       {
5087         #1 , #3 ,
5088         down = \exp_not:n { #4 } ,
5089         up = \exp_not:n { #5 } ,
5090         middle = \exp_not:n { #6 }
5091       }
5092     }
5093   }
5094 }

```

```

5095 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5096 {
5097   \bool_set_false:N \l_@@_initial_open_bool
5098   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5099   \int_set:Nn \l_@@_initial_j_int { #2 }
5100   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5101   \int_compare:nNnTF { #1 } = 1
5102   {
5103     \int_set:Nn \l_@@_initial_i_int 1
5104     \bool_set_true:N \l_@@_initial_open_bool
5105   }
5106   {
5107     \cs_if_exist:cTF
5108     {
5109       pgf @ sh @ ns @ \@@_env:
5110       - \int_eval:n { #1 - 1 }
5111       - \int_use:N \l_@@_initial_j_int
5112     }
5113     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5114     {
5115       \int_set:Nn \l_@@_initial_i_int { #1 }
5116       \bool_set_true:N \l_@@_initial_open_bool
5117     }
5118   }
5119   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5120   {
5121     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5122     \bool_set_true:N \l_@@_final_open_bool
5123   }
5124   {
5125     \cs_if_exist:cTF
5126     {
5127       pgf @ sh @ ns @ \@@_env:
5128       - \int_eval:n { #1 + #3 }
5129       - \int_use:N \l_@@_final_j_int
5130     }
5131     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5132     {
5133       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5134       \bool_set_true:N \l_@@_final_open_bool
5135     }
5136   }
5137   \group_begin:
5138   \@@_open_shorten:
5139   \int_if_zero:nTF { #2 }
5140   { \color { nicematrix-first-col } }
5141   {
5142     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5143     { \color { nicematrix-last-col } }
5144   }
5145   \keys_set:nn { NiceMatrix / xdots } { #4 }
5146   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5147   \@@_actually_draw_Vdots:
5148   \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).

```

5149   \int_step_inline:nnn { #1 } { #1 + #3 - 1 }

```

```

5150      { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5151    }

```

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

```

5152 \NewDocumentCommand \@@_rotate: { 0 { } }
5153 {
5154   \peek_remove_spaces:n
5155   {
5156     \bool_gset_true:N \g_@@_rotate_bool
5157     \keys_set:nn { NiceMatrix / rotate } { #1 }
5158   }
5159 }

5160 \keys_define:nn { NiceMatrix / rotate }
5161 {
5162   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5163   c .value_forbidden:n = true ,
5164   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5165 }

```

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

```

5166 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5167 {
5168   \tl_if_empty:nTF { #2 }
5169   { #1 }
5170   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5171 }
5172 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5173 { \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).

```

5174 \hook_gput_code:nnn { begindocument } { . }
5175 {
5176   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5177   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5178   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
5179   {

```

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

```

5180 \group_begin:
5181 \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5182 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5183 \use:e
5184 {
5185 \@@_line_i:nn
5186 { \@@_double_int_eval:n #2 - \q_stop }
5187 { \@@_double_int_eval:n #3 - \q_stop }
5188 }
5189 \group_end:
5190 }
5191 }

5192 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5193 {
5194 \bool_set_false:N \l_@@_initial_open_bool
5195 \bool_set_false:N \l_@@_final_open_bool
5196 \bool_if:nTF
5197 {
5198 \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
5199 ||
5200 \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
5201 }
5202 {
5203 \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
5204 }

```

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

```

5205 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5206 }

5207 \hook_gput_code:nnn { begindocument } { . }
5208 {
5209 \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5210 {

```

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:.`

```

5211 \c_@@_pgfortikzpicture_tl
5212 \@@_draw_line_iii:nn { #1 } { #2 }
5213 \c_@@_endpgfortikzpicture_tl
5214 }
5215 }

```

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

```

5216 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5217 {
5218 \pgfrememberpicturepositiononpagetrue
5219 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5220 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5221 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5222 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5223 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5224 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5225 \@@_draw_line:
5226 }

```

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

```

5227 \keys_define:nn { NiceMatrix / RowStyle }
5228 {
5229   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5230   cell-space-top-limit .initial:n = \c_zero_dim ,
5231   cell-space-top-limit .value_required:n = true ,
5232   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5233   cell-space-bottom-limit .initial:n = \c_zero_dim ,
5234   cell-space-bottom-limit .value_required:n = true ,
5235   cell-space-limits .meta:n =
5236     {
5237       cell-space-top-limit = #1 ,
5238       cell-space-bottom-limit = #1 ,
5239     } ,
5240   color .tl_set:N = \l_@@_color_tl ,
5241   color .value_required:n = true ,
5242   bold .bool_set:N = \l_tmpa_bool ,
5243   bold .default:n = true ,
5244   bold .initial:n = false ,
5245   nb-rows .code:n =
5246     \str_if_eq:nnTF { #1 } { * }
5247     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5248     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5249   nb-rows .value_required:n = true ,
5250   rowcolor .tl_set:N = \l_tmpa_tl ,
5251   rowcolor .value_required:n = true ,
5252   rowcolor .initial:n = ,
5253   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
5254 }

```

```

5255 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5256 {
5257   \group_begin:
5258   \tl_clear:N \l_tmpa_tl % value of \rowcolor
5259   \tl_clear:N \l_@@_color_tl
5260   \int_set:Nn \l_@@_key_nb_rows_int 1
5261   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5262   \tl_if_empty:NF \l_tmpa_tl
5263   {

```

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

```

5264     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5265     {

```

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

```

5266       \@@_exp_color_arg:N \@@_rectanglecolor \l_tmpa_tl
5267       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5268       { \int_use:N \c@iRow - * }
5269     }

```

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

```

5270     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
5271     {
5272       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5273       {
5274         \@@_exp_color_arg:N \@@_rowcolor \l_tmpa_tl
5275         {
5276           \int_eval:n { \c@iRow + 1 }
5277           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5278         }
5279       }

```

```

5280     }
5281   }
5282   \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
5283   \tl_gput_right:Nx \g_@@_row_style_tl
5284   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5285   \tl_gput_right:Nn \g_@@_row_style_tl { #2 }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
5286   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5287   {
5288     \tl_gput_right:Nx \g_@@_row_style_tl
5289     {
5290       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5291       {
5292         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5293         { \dim_use:N \l_tmpa_dim }
5294       }
5295     }
5296   }
\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
5297   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5298   {
5299     \tl_gput_right:Nx \g_@@_row_style_tl
5300     {
5301       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5302       {
5303         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5304         { \dim_use:N \l_tmpb_dim }
5305       }
5306     }
5307   }
\l_@@_color_tl is the value of the key color of \RowStyle.
5308   \tl_if_empty:NF \l_@@_color_tl
5309   {
5310     \tl_gput_right:Nx \g_@@_row_style_tl
5311     {
5312       \mode_leave_vertical:
5313       \@@_color:n { \l_@@_color_tl }
5314     }
5315   }
\l_tmpa_bool is the value of the key bold.
5316   \bool_if:NT \l_tmpa_bool
5317   {
5318     \tl_gput_right:Nn \g_@@_row_style_tl
5319     {
5320       \if_mode_math:
5321         \c_math_toggle_token
5322         \bfseries \boldmath
5323         \c_math_toggle_token
5324       \else:
5325         \bfseries \boldmath
5326       \fi:
5327     }
5328   }
5329   \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5330   \group_end:
5331   \g_@@_row_style_tl
5332   \ignorespaces
5333 }

```

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).

```
5334 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5335 {
```

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.

```
5336 \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`.

```
5337 \str_if_in:nnF { #1 } { !! }
5338 {
5339 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5340 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5341 }
5342 \int_if_zero:nTF \l_tmpa_int
```

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

```
5343 {
5344 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5345 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5346 }
```

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

```
5347 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5348 }
```

```
5349 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5350 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5351 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5352 {
5353 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5354 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5355     \group_begin:
5356     \pgfsetcornersarced
5357     {
5358         \pgfpoint
5359         { \l_@@_tab_rounded_corners_dim }
5360         { \l_@@_tab_rounded_corners_dim }
5361     }

```

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.

```

5362     \bool_if:NTF \l_@@_hvlines_bool
5363     {
5364         \pgfpathrectanglecorners
5365         {
5366             \pgfpointadd
5367             { \@@_qpoint:n { row-1 } }
5368             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5369         }
5370         {
5371             \pgfpointadd
5372             {
5373                 \@@_qpoint:n
5374                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5375             }
5376             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5377         }
5378     }
5379     {
5380         \pgfpathrectanglecorners
5381         { \@@_qpoint:n { row-1 } }
5382         {
5383             \pgfpointadd
5384             {
5385                 \@@_qpoint:n
5386                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5387             }
5388             { \pgfpoint \c_zero_dim \arrayrulewidth }
5389         }
5390     }
5391     \pgfusepath { clip }
5392     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5393     }
5394 }

```

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`).

```

5395 \cs_new_protected:Npn \@@_actually_color:
5396 {
5397     \pgfpicture
5398     \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.

```

5399     \@@_clip_with_rounded_corners:
5400     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5401     {
5402         \begin { pgfscope }

```



```

5403         \@@_color_opacity ##2
5404         \use:c { g_@@_color _ ##1 _t1 }
5405         \tl_gclear:c { g_@@_color _ ##1 _t1 }
5406         \pgfusepath { fill }
5407     \end { pgfscope }
5408 }
5409 \endpgfpicture
5410 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5411 \cs_new_protected:Npn \@@_color_opacity
5412 {
5413     \peek_meaning:NTF [
5414         { \@@_color_opacity:w }
5415         { \@@_color_opacity:w [ ] }
5416     }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5417 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5418 {
5419     \tl_clear:N \l_tmpa_tl
5420     \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.

```

5421     \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5422     \tl_if_empty:NTF \l_tmpb_tl
5423         { \@declaredcolor }
5424         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5425 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5426 \keys_define:nn { nicematrix / color-opacity }
5427 {
5428     opacity .tl_set:N          = \l_tmpa_tl ,
5429     opacity .value_required:n = true
5430 }

5431 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5432 {
5433     \tl_set:Nn \l_@@_rows_tl { #1 }
5434     \tl_set:Nn \l_@@_cols_tl { #2 }
5435     \@@_cartesian_path:
5436 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5437 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5438 {
5439     \tl_if_blank:nF { #2 }
5440     {
5441         \@@_add_to_colors_seq:en
5442         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5443         { \@@_cartesian_color:nn { #3 } { - } }
5444     }
5445 }

```

Here an example : \@@_columncolor:nn {red!15} {1,3,5-7,10-}

```

5446 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5447 {
5448   \tl_if_blank:nF { #2 }
5449   {
5450     \@@_add_to_colors_seq:en
5451     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5452     { \@@_cartesian_color:nn { - } { #3 } }
5453   }
5454 }

```

Here is an example : \@@_rectanglecolor{red!15}{2-3}{5-6}

```

5455 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5456 {
5457   \tl_if_blank:nF { #2 }
5458   {
5459     \@@_add_to_colors_seq:en
5460     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5461     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5462   }
5463 }

```

The last argument is the radius of the corners of the rectangle.

```

5464 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5465 {
5466   \tl_if_blank:nF { #2 }
5467   {
5468     \@@_add_to_colors_seq:en
5469     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5470     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5471   }
5472 }

```

The last argument is the radius of the corners of the rectangle.

```

5473 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5474 {
5475   \@@_cut_on_hyphen:w #1 \q_stop
5476   \tl_clear_new:N \l_@@_tmpc_tl
5477   \tl_clear_new:N \l_@@_tmpd_tl
5478   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5479   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5480   \@@_cut_on_hyphen:w #2 \q_stop
5481   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5482   \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.

```

5483   \@@_cartesian_path:n { #3 }
5484 }

```

Here is an example : \@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}

```

5485 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5486 {
5487   \clist_map_inline:nn { #3 }
5488   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5489 }

```

```

5490 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5491 {
5492   \int_step_inline:nn { \int_use:N \c@iRow }

```

```

5493 {
5494   \int_step_inline:nn { \int_use:N \c@jCol }
5495   {
5496     \int_if_even:nTF { ####1 + ##1 }
5497       { \@@_cellcolor [ #1 ] { #2 } }
5498       { \@@_cellcolor [ #1 ] { #3 } }
5499     { ##1 - ####1 }
5500   }
5501 }
5502 }

```

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”.

```

5503 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5504 {
5505   \@@_rectanglecolor [ #1 ] { #2 }
5506   { 1 - 1 }
5507   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5508 }

5509 \keys_define:nn { NiceMatrix / rowcolors }
5510 {
5511   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5512   respect-blocks .default:n = true ,
5513   cols .tl_set:N = \l_@@_cols_tl ,
5514   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5515   restart .default:n = true ,
5516   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5517 }

```

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*.

```

5518 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5519 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5520   \group_begin:
5521   \seq_clear_new:N \l_@@_colors_seq
5522   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5523   \tl_clear_new:N \l_@@_cols_tl
5524   \tl_set:Nn \l_@@_cols_tl { - }
5525   \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).

```

5526   \int_zero_new:N \l_@@_color_int
5527   \int_set:Nn \l_@@_color_int 1
5528   \bool_if:NT \l_@@_respect_blocks_bool
5529   {

```

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`).

```

5530   \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5531   \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5532   { \@@_not_in_exterior_p:nnnnn ##1 }
5533 }

```

```

5534 \pgfpicture
5535 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5536 \clist_map_inline:nn { #2 }
5537 {
5538   \tl_set:Nn \l_tmpa_tl { ##1 }
5539   \tl_if_in:NnTF \l_tmpa_tl { - }
5540   { \@@_cut_on_hyphen:w ##1 \q_stop }
5541   { \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.

```

5542   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5543   \int_set:Nn \l_@@_color_int
5544   { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5545   \int_zero_new:N \l_@@_tmpc_int
5546   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5547   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5548   {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5549       \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).

```

5550       \bool_if:NT \l_@@_respect_blocks_bool
5551       {
5552         \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5553         { \@@_intersect_our_row_p:nnnnn ###1 }
5554         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5555       }
5556       \tl_set:Nx \l_@@_rows_tl
5557       { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5558       \tl_clear_new:N \l_@@_color_tl
5559       \tl_set:Nx \l_@@_color_tl
5560       {
5561         \@@_color_index:n
5562         {
5563           \int_mod:nn
5564             { \l_@@_color_int - 1 }
5565             { \seq_count:N \l_@@_colors_seq }
5566           + 1
5567         }
5568       }
5569       \tl_if_empty:NF \l_@@_color_tl
5570       {
5571         \@@_add_to_colors_seq:ee
5572         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5573         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5574       }
5575       \int_incr:N \l_@@_color_int
5576       \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5577     }
5578   }
5579 \endpgfpicture
5580 \group_end:
5581 }

```

The command `\@@_color_index:n` peaks 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.

```

5582 \cs_new:Npn \@@_color_index:n #1
5583 {

```

```

5584 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5585 { \@@_color_index:n { #1 - 1 } }
5586 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5587 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5588 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5589 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5590 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5591 {
5592   \int_compare:nNnT { #3 } > \l_tmpb_int
5593   { \int_set:Nn \l_tmpb_int { #3 } }
5594 }

5595 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5596 {
5597   \bool_lazy_or:nnTF
5598   { \int_if_zero_p:n { #4 } }
5599   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5600   \prg_return_false:
5601   \prg_return_true:
5602 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5603 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5604 {
5605   \bool_if:nTF
5606   {
5607     \int_compare_p:n { #1 <= \l_tmpa_int }
5608     &&
5609     \int_compare_p:n { \l_tmpa_int <= #3 }
5610   }
5611   \prg_return_true:
5612   \prg_return_false:
5613 }

```

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`).

```

5614 \cs_new_protected:Npn \@@_cartesian_path:n #1
5615 {
5616   \bool_lazy_and:nnT
5617   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5618   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5619   {
5620     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5621     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5622   }

```

We begin the loop over the columns.

```

5623 \clist_map_inline:Nn \l_@@_cols_tl
5624 {
5625   \tl_set:Nn \l_tmpa_tl { ##1 }
5626   \tl_if_in:NnTF \l_tmpa_tl { - }
5627     { \@@_cut_on_hyphen:w ##1 \q_stop }
5628     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5629   \bool_lazy_or:nnT
5630     { \tl_if_blank_p:V \l_tmpa_tl }
5631     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5632     { \tl_set:Nn \l_tmpa_tl { 1 } }
5633   \bool_lazy_or:nnT
5634     { \tl_if_blank_p:V \l_tmpb_tl }
5635     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5636     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5637   \int_compare:nNnT \l_tmpb_tl > \c@jCol
5638     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5639   \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.

```

5640   \@@_qpoint:n { col - \l_tmpa_tl }
5641   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5642     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5643     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5644   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5645   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5646 \clist_map_inline:Nn \l_@@_rows_tl
5647 {
5648   \tl_set:Nn \l_tmpa_tl { ####1 }
5649   \tl_if_in:NnTF \l_tmpa_tl { - }
5650     { \@@_cut_on_hyphen:w ####1 \q_stop }
5651     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5652   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5653   \tl_if_empty:NT \l_tmpb_tl
5654     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5655   \int_compare:nNnT \l_tmpb_tl > \c@iRow
5656     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```

5657   \seq_if_in:NxF \l_@@_corners_cells_seq
5658     { \l_tmpa_tl - \l_@@_tmpc_tl }
5659     {
5660       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5661       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5662       \@@_qpoint:n { row - \l_tmpa_tl }
5663       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5664       \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5665       \pgfpathrectanglecorners
5666         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5667         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5668     }
5669   }
5670 }
5671 }

```

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`).

```

5672 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-* and `\c@jCol` equal to 10, the clist `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

5673 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5674 {
5675   \clist_set_eq:NN \l_tmpa_clist #1
5676   \clist_clear:N #1
5677   \clist_map_inline:Nn \l_tmpa_clist
5678   {
5679     \tl_set:Nn \l_tmpa_tl { ##1 }
5680     \tl_if_in:NnTF \l_tmpa_tl { - }
5681     { \@@_cut_on_hyphen:w ##1 \q_stop }
5682     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5683     \bool_lazy_or:nnT
5684     { \tl_if_blank_p:V \l_tmpa_tl }
5685     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5686     { \tl_set:Nn \l_tmpa_tl { 1 } }
5687     \bool_lazy_or:nnT
5688     { \tl_if_blank_p:V \l_tmpb_tl }
5689     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5690     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5691     \int_compare:nNnT \l_tmpb_tl > #2
5692     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5693     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5694     { \clist_put_right:Nn #1 { ####1 } }
5695   }
5696 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5697 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5698 {
5699   \@@_test_color_inside:
5700   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5701   {

```

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`).

```

5702     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5703     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5704   }
5705   \ignorespaces
5706 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5707 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5708 {
5709   \@@_test_color_inside:
5710   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5711   {
5712     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5713     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5714     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5715   }
5716   \ignorespaces
5717 }

```

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).

```

5718 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5719 { \@@_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.

```

5720 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5721 {
5722   \@@_test_color_inside:
5723   \peek_remove_spaces:n
5724   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5725 }

5726 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5727 {

```

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`.

```

5728   \seq_gclear:N \g_tmpa_seq
5729   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5730   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5731   \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`.

```

5732   \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5733   {
5734     { \int_use:N \c@iRow }
5735     { \exp_not:n { #1 } }
5736     { \exp_not:n { #2 } }
5737     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5738   }
5739 }

```

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`).

```

5740 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5741 {
5742   \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).

```

5743   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5744   {
5745     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5746     {
5747       \@@_rowlistcolors
5748       [ \exp_not:n { #2 } ]
5749       { #1 - \int_eval:n { \c@iRow - 1 } }
5750       { \exp_not:n { #3 } }
5751       [ \exp_not:n { #4 } ]
5752     }
5753   }
5754 }

```


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.

```

5755 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5756 {
5757   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5758     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5759   \seq_gclear:N \g_@@_rowlistcolors_seq
5760 }

5761 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5762 {
5763   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5764     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5765 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5766 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5767 {

```

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).

```

5768   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5769   {

```

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).

```

5770     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5771     {
5772       \exp_not:N \columncolor [ #1 ]
5773       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5774     }
5775   }
5776 }

```

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`.

```

5777 \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`.

```

5778 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5779 {
5780   \int_if_zero:nTF \l_@@_first_col_int
5781     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5782     {
5783       \int_if_zero:nTF \c@jCol
5784         {
5785           \int_compare:nNnF \c@iRow = { -1 }
5786             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5787         }
5788       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5789     }
5790 }

```

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.

```

5791 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5792 {
5793   \int_if_zero:nF \c@iRow
5794   {
5795     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5796     {
5797       \int_compare:nNnT \c@jCol > \c_zero_int
5798         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5799     }
5800   }
5801 }

```

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.

```

5802 \keys_define:nn { NiceMatrix / Rules }
5803 {
5804   position .int_set:N = \l_@@_position_int ,
5805   position .value_required:n = true ,
5806   start .int_set:N = \l_@@_start_int ,
5807   start .initial:n = 1 ,
5808   end .code:n =
5809     \bool_lazy_or:nnTF
5810       { \tl_if_empty_p:n { #1 } }
5811       { \str_if_eq_p:nn { #1 } { last } }
5812       { \int_set_eq:NN \l_@@_end_int \c@jCol }
5813       { \int_set:Nn \l_@@_end_int { #1 } }
5814 }

```

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.

```

5815 \keys_define:nn { NiceMatrix / RulesBis }
5816 {
5817   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5818   multiplicity .initial:n = 1 ,
5819   dotted .bool_set:N = \l_@@_dotted_bool ,
5820   dotted .initial:n = false ,
5821   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]`.

```

5822   color .code:n =
5823     \@@_set_CT@arc@:n { #1 }
5824     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5825   color .value_required:n = true ,
5826   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5827   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.

```

5828   tikz .code:n =
5829     \IfPackageLoadedTF { tikz }
5830       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5831       { \@@_error:n { tikz-without-tikz } } ,
5832   tikz .value_required:n = true ,
5833   total-width .dim_set:N = \l_@@_rule_width_dim ,
5834   total-width .value_required:n = true ,
5835   width .meta:n = { total-width = #1 } ,
5836   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5837 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5838 \cs_new_protected:Npn \@@_vline:n #1
5839 {

```

The group is for the options.

```

5840   \group_begin:
5841   \int_zero_new:N \l_@@_end_int
5842   \int_set_eq:NN \l_@@_end_int \c@iRow
5843   \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).

```

5844   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5845     \@@_vline_i:
5846   \group_end:
5847 }

```

```

5848 \cs_new_protected:Npn \@@_vline_i:
5849 {
5850   \int_zero_new:N \l_@@_local_start_int
5851   \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`.

```

5852   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5853   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5854     \l_tmpa_tl
5855   {

```

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.

```

5856     \bool_gset_true:N \g_tmpa_bool
5857     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5858       { \@@_test_vline_in_block:nnnnn ##1 }
5859     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5860       { \@@_test_vline_in_block:nnnnn ##1 }
5861     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5862       { \@@_test_vline_in_stroken_block:nnnn ##1 }
5863     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5864     \bool_if:NTF \g_tmpa_bool
5865       {
5866         \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.

```

5867         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5868       }
5869     {
5870       \int_compare:nNnT \l_@@_local_start_int > 0
5871       {
5872         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5873         \@@_vline_ii:
5874         \int_zero:N \l_@@_local_start_int
5875       }
5876     }
5877   }
5878   \int_compare:nNnT \l_@@_local_start_int > 0
5879   {
5880     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5881     \@@_vline_ii:
5882   }
5883 }

```

```

5884 \cs_new_protected:Npn \@@_test_in_corner_v:
5885 {
5886   \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5887   {
5888     \seq_if_in:NxT
5889       \l_@@_corners_cells_seq
5890       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5891     { \bool_set_false:N \g_tmpa_bool }
5892   }
5893   {
5894     \seq_if_in:NxT
5895       \l_@@_corners_cells_seq
5896       { \l_tmpa_tl - \l_tmpb_tl }
5897     {
5898       \int_compare:nNnTF \l_tmpb_tl = 1
5899       { \bool_set_false:N \g_tmpa_bool }
5900       {
5901         \seq_if_in:NxT
5902           \l_@@_corners_cells_seq
5903           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5904         { \bool_set_false:N \g_tmpa_bool }
5905       }
5906     }
5907   }
5908 }

```

```

5909 \cs_new_protected:Npn \@@_vline_ii:
5910 {
5911   \tl_clear:N \l_@@_tikz_rule_tl
5912   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5913   \bool_if:NTF \l_@@_dotted_bool
5914     \@@_vline_iv:
5915     {
5916       \tl_if_empty:NTF \l_@@_tikz_rule_tl
5917         \@@_vline_iii:
5918         \@@_vline_v:
5919     }
5920 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

5921 \cs_new_protected:Npn \@@_vline_iii:
5922 {
5923   \pgfpicture
5924   \pgfrememberpicturepositiononpagetrue
5925   \pgf@relevantforpicturesizefalse
5926   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5927   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5928   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5929   \dim_set:Nn \l_tmpb_dim
5930   {
5931     \pgf@x
5932     - 0.5 \l_@@_rule_width_dim
5933     +
5934     ( \arrayrulewidth * \l_@@_multiplicity_int
5935       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5936   }
5937   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5938   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5939   \bool_lazy_all:nT
5940   {
5941     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5942     { \cs_if_exist_p:N \CT@drsc@ }
5943     { ! \tl_if_blank_p:V \CT@drsc@ }
5944   }
5945   {
5946     \group_begin:
5947     \CT@drsc@
5948     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5949     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5950     \dim_set:Nn \l_@@_tmpd_dim
5951     {
5952       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5953       * ( \l_@@_multiplicity_int - 1 )
5954     }
5955     \pgfpathrectanglecorners
5956     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5957     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5958     \pgfusepath { fill }
5959     \group_end:
5960   }
5961   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5962   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5963   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5964   {
5965     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5966     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5967     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5968     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5969   }

```

```

5970 \CT@arc@
5971 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5972 \pgfsetrectcap
5973 \pgfusepathqstroke
5974 \endpgfpicture
5975 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

5976 \cs_new_protected:Npn \@@_vline_iv:
5977 {
5978   \pgfpicture
5979   \pgfrememberpicturepositiononpagetrue
5980   \pgf@relevantforpicturesizefalse
5981   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5982   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5983   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5984   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5985   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5986   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5987   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5988   \CT@arc@
5989   \@@_draw_line:
5990   \endpgfpicture
5991 }

```

The following code is for the case when the user uses the key `tikz`.

```

5992 \cs_new_protected:Npn \@@_vline_v:
5993 {
5994   \begin {tikzpicture }
5995   % 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).

```

5996 \CT@arc@
5997 \tl_if_empty:NF \l_@@_rule_color_tl
5998 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
5999 \pgfrememberpicturepositiononpagetrue
6000 \pgf@relevantforpicturesizefalse
6001 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6002 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6003 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6004 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6005 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6006 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6007 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6008 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6009 ( \l_tmpb_dim , \l_tmpa_dim ) --
6010 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6011 \end { tikzpicture }
6012 }

```

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).

```

6013 \cs_new_protected:Npn \@@_draw_vlines:
6014 {
6015   \int_step_inline:nnn
6016   {
6017     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6018     1 2
6019   }
6020   {

```

```

6021     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6022     { \int_eval:n { \c@jCol + 1 } }
6023     \c@jCol
6024   }
6025   {
6026     \tl_if_eq:NnF \l_@@_vlines_clist { all }
6027     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6028     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6029   }
6030 }

```

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}`.

```

6031 \cs_new_protected:Npn \@@_hline:n #1
6032 {

```

The group is for the options.

```

6033   \group_begin:
6034   \int_zero_new:N \l_@@_end_int
6035   \int_set_eq:NN \l_@@_end_int \c@jCol
6036   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6037   \@@_hline_i:
6038   \group_end:
6039 }

6040 \cs_new_protected:Npn \@@_hline_i:
6041 {
6042   \int_zero_new:N \l_@@_local_start_int
6043   \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`.

```

6044   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
6045   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6046     \l_tmpb_tl
6047   {

```

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.

```

6048     \bool_gset_true:N \g_tmpa_bool
6049     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6050       { \@@_test_hline_in_block:nnnnn ##1 }
6051     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6052       { \@@_test_hline_in_block:nnnnn ##1 }
6053     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6054       { \@@_test_hline_in_stroken_block:nnnn ##1 }
6055     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6056     \bool_if:NTF \g_tmpa_bool
6057     {
6058       \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.

```

6059       { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6060     }
6061     {
6062       \int_compare:nNnT \l_@@_local_start_int > 0
6063       {
6064         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6065         \@@_hline_ii:

```

```

6066         \int_zero:N \l_@@_local_start_int
6067     }
6068 }
6069 }
6070 \int_compare:nNnT \l_@@_local_start_int > 0
6071 {
6072     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6073     \@@_hline_ii:
6074 }
6075 }

6076 \cs_new_protected:Npn \@@_test_in_corner_h:
6077 {
6078     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6079     {
6080         \seq_if_in:NxT
6081             \l_@@_corners_cells_seq
6082             { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6083             { \bool_set_false:N \g_tmpa_bool }
6084     }
6085     {
6086         \seq_if_in:NxT
6087             \l_@@_corners_cells_seq
6088             { \l_tmpa_tl - \l_tmpb_tl }
6089         {
6090             \int_compare:nNnTF \l_tmpa_tl = 1
6091             { \bool_set_false:N \g_tmpa_bool }
6092             {
6093                 \seq_if_in:NxT
6094                     \l_@@_corners_cells_seq
6095                     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6096                     { \bool_set_false:N \g_tmpa_bool }
6097             }
6098         }
6099     }
6100 }

6101 \cs_new_protected:Npn \@@_hline_ii:
6102 {
6103     \tl_clear:N \l_@@_tikz_rule_tl
6104     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6105     \bool_if:NTF \l_@@_dotted_bool
6106         \@@_hline_iv:
6107     {
6108         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6109         \@@_hline_iii:
6110         \@@_hline_v:
6111     }
6112 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6113 \cs_new_protected:Npn \@@_hline_iii:
6114 {
6115     \pgfpicture
6116     \pgfrememberpicturepositiononpagetrue
6117     \pgf@relevantforpicturesizefalse
6118     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6119     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6120     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6121     \dim_set:Nn \l_tmpb_dim
6122     {

```



```

6123     \pgf@y
6124     - 0.5 \l_@@_rule_width_dim
6125     +
6126     ( \arrayrulewidth * \l_@@_multiplicity_int
6127       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6128   }
6129   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6130   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6131   \bool_lazy_all:nT
6132   {
6133     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6134     { \cs_if_exist_p:N \CT@drsc@ }
6135     { ! \tl_if_blank_p:V \CT@drsc@ }
6136   }
6137   {
6138     \group_begin:
6139     \CT@drsc@
6140     \dim_set:Nn \l_@@_tmpd_dim
6141     {
6142       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6143       * ( \l_@@_multiplicity_int - 1 )
6144     }
6145     \pgfpathrectanglecorners
6146     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6147     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6148     \pgfusepathqfill
6149     \group_end:
6150   }
6151   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6152   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6153   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6154   {
6155     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6156     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6157     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6158     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6159   }
6160   \CT@arc@
6161   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6162   \pgfsetrectcap
6163   \pgfusepathqstroke
6164   \endpgfpicture
6165 }

```

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6166 \cs_new_protected:Npn \@@_hline_iv:
6167 {
6168   \pgfpicture
6169   \pgfrememberpicturepositiononpagetrue
6170   \pgf@relevantforpicturesizefalse
6171   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6172   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6173   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6174   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6175   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6176   \int_compare:nNnT \l_@@_local_start_int = 1
6177   {
6178     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6179     \bool_if:NF \g_@@_delims_bool
6180     { \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.

```

6181     \tl_if_eq:NnF \g_@@_left_delim_tl (
6182       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6183     )
6184     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6185     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6186     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6187     {
6188       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6189       \bool_if:NF \g_@@_delims_bool
6190       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6191       \tl_if_eq:NnF \g_@@_right_delim_tl )
6192       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6193     }
6194     \CT@arc@
6195     \@@_draw_line:
6196     \endpgfpicture
6197   }

```

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`).

```

6198 \cs_new_protected:Npn \@@_hline_v:
6199 {
6200   \begin { tikzpicture }
6201   % 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).

```

6202   \CT@arc@
6203   \tl_if_empty:NF \l_@@_rule_color_tl
6204   { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6205   \pgfrememberpicturepositiononpagetrue
6206   \pgf@relevantforpicturesizefalse
6207   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6208   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6209   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6210   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6211   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6212   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6213   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6214   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6215   ( \l_tmpa_dim , \l_tmpb_dim ) --
6216   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6217   \end { tikzpicture }
6218 }

```

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).

```

6219 \cs_new_protected:Npn \@@_draw_hlines:
6220 {
6221   \int_step_inline:nnn
6222     {
6223       \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6224         1 2
6225     }
6226     {
6227       \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6228         { \int_eval:n { \c@iRow + 1 } }
6229         \c@iRow
6230     }
6231     {
6232       \tl_if_eq:NnF \l_@@_hlines_clist { all }
6233         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6234         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6235     }
6236 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6237 \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.

```

6238 \cs_set:Npn \@@_Hline_i:n #1
6239 {
6240   \peek_remove_spaces:n
6241     {
6242       \peek_meaning:NTF \Hline
6243         { \@@_Hline_ii:nn { #1 + 1 } }
6244         { \@@_Hline_iii:n { #1 } }
6245     }
6246 }
6247 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6248 \cs_set:Npn \@@_Hline_iii:n #1
6249 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6250 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6251 {
6252   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6253   \skip_vertical:n { \l_@@_rule_width_dim }
6254   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6255     {
6256       \@@_hline:n
6257         {
6258           multiplicity = #1 ,
6259           position = \int_eval:n { \c@iRow + 1 } ,
6260           total-width = \dim_use:N \l_@@_rule_width_dim ,
6261           #2
6262         }
6263     }
6264   \egroup
6265 }

```

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`).

```

6266 \cs_new_protected:Npn \l_@@_custom_line:n #1
6267 {
6268   \str_clear_new:N \l_@@_command_str
6269   \str_clear_new:N \l_@@_ccommand_str
6270   \str_clear_new:N \l_@@_letter_str
6271   \tl_clear_new:N \l_@@_other_keys_tl
6272   \keys_set_known:nn { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6273   \bool_lazy_all:nTF
6274   {
6275     { \str_if_empty_p:N \l_@@_letter_str }
6276     { \str_if_empty_p:N \l_@@_command_str }
6277     { \str_if_empty_p:N \l_@@_ccommand_str }
6278   }
6279   { \@@_error:n { No~letter~and~no~command } }
6280   { \exp_args:NV \l_@@_custom_line_i:n \l_@@_other_keys_tl }
6281 }
6282 \keys_define:nn { NiceMatrix / custom-line }
6283 {
6284   letter .str_set:N = \l_@@_letter_str ,
6285   letter .value_required:n = true ,
6286   command .str_set:N = \l_@@_command_str ,
6287   command .value_required:n = true ,
6288   ccommand .str_set:N = \l_@@_ccommand_str ,
6289   ccommand .value_required:n = true ,
6290 }

```

```

6291 \cs_new_protected:Npn \l_@@_custom_line_i:n #1
6292 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6293   \bool_set_false:N \l_@@_tikz_rule_bool
6294   \bool_set_false:N \l_@@_dotted_rule_bool
6295   \bool_set_false:N \l_@@_color_bool
6296   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6297   \bool_if:NT \l_@@_tikz_rule_bool
6298   {
6299     \IfPackageLoadedTF { tikz }
6300     { }
6301     { \@@_error:n { tikz~in~custom~line~without~tikz } }
6302     \bool_if:NT \l_@@_color_bool
6303     { \@@_error:n { color~in~custom~line~with~tikz } }
6304   }
6305   \bool_if:nT
6306   {
6307     \int_compare_p:nNn \l_@@_multiplicity_int > 1
6308     && \l_@@_dotted_rule_bool
6309   }
6310   { \@@_error:n { key~multiplicity~with~dotted } }
6311   \str_if_empty:NF \l_@@_letter_str
6312   {
6313     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6314     { \@@_error:n { Several~letters } }
6315     {
6316       \exp_args:NnV \tl_if_in:NnTF
6317       \c_@@_forbidden_letters_str \l_@@_letter_str
6318       { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6319       {

```

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.

```

6320      \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6321      { \@@_v_custom_line:n { #1 } }
6322    }
6323  }
6324 }
6325 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6326 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6327 }

6328 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6329 \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.

```

6330 \keys_define:nn { NiceMatrix / custom-line-bis }
6331 {
6332   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6333   multiplicity .initial:n = 1 ,
6334   multiplicity .value_required:n = true ,
6335   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6336   color .value_required:n = true ,
6337   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6338   tikz .value_required:n = true ,
6339   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6340   dotted .value_forbidden:n = true ,
6341   total-width .code:n = { } ,
6342   total-width .value_required:n = true ,
6343   width .code:n = { } ,
6344   width .value_required:n = true ,
6345   sep-color .code:n = { } ,
6346   sep-color .value_required:n = true ,
6347   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6348 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6349 \bool_new:N \l_@@_dotted_rule_bool
6350 \bool_new:N \l_@@_tikz_rule_bool
6351 \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`.

```

6352 \keys_define:nn { NiceMatrix / custom-line-width }
6353 {
6354   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6355   multiplicity .initial:n = 1 ,
6356   multiplicity .value_required:n = true ,
6357   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6358   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6359       \bool_set_true:N \l_@@_total_width_bool ,
6360   total-width .value_required:n = true ,
6361   width .meta:n = { total-width = #1 } ,
6362   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6363 }

```

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).

```
6364 \cs_new_protected:Npn \@@_h_custom_line:n #1
6365 {
```

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).

```
6366 \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6367 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6368 }
```

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).

```
6369 \cs_new_protected:Npn \@@_c_custom_line:n #1
6370 {
```

Here, we need an expandable command since it begins with an \noalign.

```
6371 \exp_args:Nc \NewExpandableDocumentCommand
6372 { nicematrix - \l_@@_ccommand_str }
6373 { 0 { } m }
6374 {
6375 \noalign
6376 {
6377 \@@_compute_rule_width:n { #1 , ##1 }
6378 \skip_vertical:n { \l_@@_rule_width_dim }
6379 \clist_map_inline:nn
6380 { ##2 }
6381 { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6382 }
6383 }
6384 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6385 }
```

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*.

```
6386 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6387 {
6388 \str_if_in:nnTF { #2 } { - }
6389 { \@@_cut_on_hyphen:w #2 \q_stop }
6390 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6391 \tl_gput_right:Nx \g_@@_pre_code_after_tl
6392 {
6393 \@@_hline:n
6394 {
6395 #1 ,
6396 start = \l_tmpa_tl ,
6397 end = \l_tmpb_tl ,
6398 position = \int_eval:n { \c@iRow + 1 } ,
6399 total-width = \dim_use:N \l_@@_rule_width_dim
6400 }
6401 }
6402 }
```

```
6403 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6404 {
6405 \bool_set_false:N \l_@@_tikz_rule_bool
6406 \bool_set_false:N \l_@@_total_width_bool
6407 \bool_set_false:N \l_@@_dotted_rule_bool
6408 \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6409 \bool_if:NF \l_@@_total_width_bool
```

```

6410 {
6411   \bool_if:NTF \l_@@_dotted_rule_bool
6412   { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6413   {
6414     \bool_if:NF \l_@@_tikz_rule_bool
6415     {
6416       \dim_set:Nn \l_@@_rule_width_dim
6417       {
6418         \arrayrulewidth * \l_@@_multiplicity_int
6419         + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6420       }
6421     }
6422   }
6423 }
6424 }

```

```

6425 \cs_new_protected:Npn \@@_v_custom_line:n #1
6426 {
6427   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6428 \tl_gput_right:Nx \g_@@_array_preamble_tl
6429 { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6430 \tl_gput_right:Nx \g_@@_pre_code_after_tl
6431 {
6432   \@@_vline:n
6433   {
6434     #1 ,
6435     position = \int_eval:n { \c@jCol + 1 } ,
6436     total-width = \dim_use:N \l_@@_rule_width_dim
6437   }
6438 }
6439 \@@_rec_preamble:n
6440 }
6441 \@@_custom_line:n
6442 { 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.

```

6443 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6444 {
6445   \bool_lazy_all:nT
6446   {
6447     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6448     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6449     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6450     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6451   }
6452   { \bool_gset_false:N \g_tmpa_bool }
6453 }

```

The same for vertical rules.

```

6454 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6455 {
6456   \bool_lazy_all:nT
6457   {
6458     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6459     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6460     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6461     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }

```

```

6462     }
6463     { \bool_gset_false:N \g_tmpa_bool }
6464 }

6465 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6466 {
6467     \bool_lazy_all:nT
6468     {
6469         {
6470             ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6471             || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6472         }
6473         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6474         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6475     }
6476     { \bool_gset_false:N \g_tmpa_bool }
6477 }

6478 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6479 {
6480     \bool_lazy_all:nT
6481     {
6482         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6483         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6484         {
6485             ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6486             || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6487         }
6488     }
6489     { \bool_gset_false:N \g_tmpa_bool }
6490 }

```

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.

```

6491 \cs_new_protected:Npn \@@_compute_corners:
6492 {

```

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.

```

6493     \seq_clear_new:N \l_@@_corners_cells_seq
6494     \clist_map_inline:Nn \l_@@_corners_clist
6495     {
6496         \str_case:nnF { ##1 }
6497         {
6498             { NW }
6499             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6500             { NE }
6501             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6502             { SW }
6503             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6504             { SE }
6505             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6506         }
6507         { \@@_error:nn { bad~corner } { ##1 } }
6508     }

```


Even if the user has used the key `corners` the list of cells in the corners may be empty.

```
6509 \seq_if_empty:NF \l_@@_corners_cells_seq
6510 {
```

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.

```
6511 \tl_build_gput_right:Nx \g_@@_aux_tl
6512 {
6513   \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6514   { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6515 }
6516 }
6517 }
```

“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.

```
6518 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6519 {
```

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.

```
6520 \bool_set_false:N \l_tmpa_bool
6521 \int_zero_new:N \l_@@_last_empty_row_int
6522 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6523 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6524 {
6525   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6526   \bool_lazy_or:nnTF
6527   {
6528     \cs_if_exist_p:c
6529     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6530   }
6531   \l_tmpb_bool
6532   { \bool_set_true:N \l_tmpa_bool }
6533   {
6534     \bool_if:NF \l_tmpa_bool
6535     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6536   }
6537 }
```

Now, you determine the last empty cell in the row of number 1.

```
6538 \bool_set_false:N \l_tmpa_bool
6539 \int_zero_new:N \l_@@_last_empty_column_int
6540 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6541 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6542 {
6543   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6544   \bool_lazy_or:nnTF
6545   \l_tmpb_bool
6546   {
```

```

6547         \cs_if_exist_p:c
6548         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6549     }
6550     { \bool_set_true:N \l_tmpa_bool }
6551     {
6552         \bool_if:NF \l_tmpa_bool
6553         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6554     }
6555 }

```

Now, we loop over the rows.

```

6556     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6557     {

```

We treat the row number ##1 with another loop.

```

6558         \bool_set_false:N \l_tmpa_bool
6559         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6560         {
6561             \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
6562             \bool_lazy_or:nnTF
6563             \l_tmpb_bool
6564             {
6565                 \cs_if_exist_p:c
6566                 { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
6567             }
6568             { \bool_set_true:N \l_tmpa_bool }
6569             {
6570                 \bool_if:NF \l_tmpa_bool
6571                 {
6572                     \int_set:Nn \l_@@_last_empty_column_int { ####1 }
6573                     \seq_put_right:Nn
6574                     \l_@@_corners_cells_seq
6575                     { ##1 - ####1 }
6576                 }
6577             }
6578         }
6579     }
6580 }

```

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`).

```

6581 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6582 {
6583     \int_set:Nn \l_tmpa_int { #1 }
6584     \int_set:Nn \l_tmpb_int { #2 }
6585     \bool_set_false:N \l_tmpb_bool
6586     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6587     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6588 }
6589 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6590 {
6591     \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6592     {
6593         \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6594         {
6595             \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6596             {
6597                 \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6598                 { \bool_set_true:N \l_tmpb_bool }
6599             }
6600         }
6601     }
6602 }

```

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.

```
6603 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
6604 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6605 {
6606   auto-columns-width .code:n =
6607   {
6608     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6609     \dim_gzero_new:N \g_@@_max_cell_width_dim
6610     \bool_set_true:N \l_@@_auto_columns_width_bool
6611   }
6612 }

6613 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6614 {
6615   \int_gincr:N \g_@@_NiceMatrixBlock_int
6616   \dim_zero:N \l_@@_columns_width_dim
6617   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6618   \bool_if:NT \l_@@_block_auto_columns_width_bool
6619   {
6620     \cs_if_exist:cT
6621     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6622     {
6623       % is \exp_args:NNe mandatory?
6624       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6625       {
6626         \use:c
6627         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6628       }
6629     }
6630   }
6631 }
```

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).

```
6632 {
6633   \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.

```
6634   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6635   {
6636     \bool_if:NT \l_@@_block_auto_columns_width_bool
6637     {
6638       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6639       \iow_shipout:Nx \@mainaux
6640       {
6641         \cs_gset:cpn
6642         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6643         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6644       }
6645       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6646     }
6647   }
6648   \ignorespacesafterend
6649 }
```

26 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```
6650 \cs_generate_variant:Nn \dim_min:nn { v n }
6651 \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).

```
6652 \cs_new_protected:Npn \@@_create_extra_nodes:
6653 {
6654   \bool_if:nTF \l_@@_medium_nodes_bool
6655   {
6656     \bool_if:nTF \l_@@_large_nodes_bool
6657     \@@_create_medium_and_large_nodes:
6658     \@@_create_medium_nodes:
6659   }
6660   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6661 }
```

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`.

```
6662 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6663 {
6664   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6665   {
6666     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6667     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6668     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6669     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6670   }
6671   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6672   {
6673     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6674     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6675     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6676     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6677   }
}
```

We begin the two nested loops over the rows and the columns of the array.

```
6678 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6679 {
6680   \int_step_variable:nnNn
6681   \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.

```

6682     {
6683         \cs_if_exist:cT
6684         { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell (i - j). They will be stored in \pgf@x and \pgf@y.

```

6685     {
6686         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6687         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6688         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6689         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6690         {
6691             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6692             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6693         }

```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell (i - j). They will be stored in \pgf@x and \pgf@y.

```

6694         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6695         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6696         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6697         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6698         {
6699             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6700             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6701         }
6702     }
6703 }
6704 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6705     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6706     {
6707         \dim_compare:nNnT
6708         { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6709         {
6710             \@@_qpoint:n { row - \@@_i: - base }
6711             \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6712             \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6713         }
6714     }
6715     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6716     {
6717         \dim_compare:nNnT
6718         { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6719         {
6720             \@@_qpoint:n { col - \@@_j: }
6721             \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@x
6722             \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@x
6723         }
6724     }
6725 }

```

Here is the command \@@_create_medium_nodes:. When this command is used, the “medium nodes” are created.

```

6726     \cs_new_protected:Npn \@@_create_medium_nodes:
6727     {
6728         \pgfpicture
6729         \pgfrememberpicturepositiononpagetrue
6730         \pgf@relevantforpicturesizefalse
6731         \@@_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”.

```
6732 \tl_set:Nn \l_@@_suffix_tl { -medium }
6733 \@@_create_nodes:
6734 \endpgfpicture
6735 }
```

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:`.

```
6736 \cs_new_protected:Npn \@@_create_large_nodes:
6737 {
6738   \pgfpicture
6739   \pgfrememberpicturepositiononpagetrue
6740   \pgf@relevantforpicturesizefalse
6741   \@@_computations_for_medium_nodes:
6742   \@@_computations_for_large_nodes:
6743   \tl_set:Nn \l_@@_suffix_tl { - large }
6744   \@@_create_nodes:
6745   \endpgfpicture
6746 }

6747 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6748 {
6749   \pgfpicture
6750   \pgfrememberpicturepositiononpagetrue
6751   \pgf@relevantforpicturesizefalse
6752   \@@_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”.

```
6753 \tl_set:Nn \l_@@_suffix_tl { - medium }
6754 \@@_create_nodes:
6755 \@@_computations_for_large_nodes:
6756 \tl_set:Nn \l_@@_suffix_tl { - large }
6757 \@@_create_nodes:
6758 \endpgfpicture
6759 }
```

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.

```
6760 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6761 {
6762   \int_set:Nn \l_@@_first_row_int 1
6763   \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`.

```
6764 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6765 {
6766   \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6767   {
6768     (
6769       \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6770       \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6771     )
6772     / 2
6773   }
```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6774     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6775     { l_@@_row_@@_i: _min_dim }
6776   }
6777   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6778   {
6779     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6780     {
6781       (
6782         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6783         \dim_use:c
6784         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6785       )
6786       / 2
6787     }
6788     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6789     { l_@@_column _ \@@_j: _ max _ dim }
6790   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6791   \dim_sub:cn
6792   { l_@@_column _ 1 _ min _ dim }
6793   \l_@@_left_margin_dim
6794   \dim_add:cn
6795   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6796   \l_@@_right_margin_dim
6797 }

```

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).

```

6798 \cs_new_protected:Npn \@@_create_nodes:
6799 {
6800   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6801   {
6802     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6803     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6804     \@@_pgf_rect_node:nnnnn
6805     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6806     { \dim_use:c { l_@@_column _ \@@_j: _min_dim } }
6807     { \dim_use:c { l_@@_row _ \@@_i: _min_dim } }
6808     { \dim_use:c { l_@@_column _ \@@_j: _max_dim } }
6809     { \dim_use:c { l_@@_row _ \@@_i: _max_dim } }
6810     \str_if_empty:NF \l_@@_name_str
6811     {
6812       \pgfnodealias
6813       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6814       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6815     }
6816   }
6817 }

```

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 .

```

6818   \seq_map_pairwise_function:NNN
6819   \g_@@_multicolumn_cells_seq
6820   \g_@@_multicolumn_sizes_seq
6821   \@@_node_for_multicolumn:nn
6822 }

```

```

6823 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6824 {
6825   \cs_set_nopar:Npn \@@_i: { #1 }
6826   \cs_set_nopar:Npn \@@_j: { #2 }
6827 }

```

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”).

```

6828 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6829 {
6830   \@@_extract_coords_values: #1 \q_stop
6831   \@@_pgf_rect_node:nnnnn
6832   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6833   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6834   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6835   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6836   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6837   \str_if_empty:NF \l_@@_name_str
6838   {
6839     \pgfnodealias
6840     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6841     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6842   }
6843 }

```

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.

```

6844 \keys_define:nn { NiceMatrix / Block / FirstPass }
6845 {
6846   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6847   l .value_forbidden:n = true ,
6848   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6849   r .value_forbidden:n = true ,
6850   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6851   c .value_forbidden:n = true ,
6852   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6853   L .value_forbidden:n = true ,
6854   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6855   R .value_forbidden:n = true ,
6856   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6857   C .value_forbidden:n = true ,
6858   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6859   t .value_forbidden:n = true ,
6860   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6861   T .value_forbidden:n = true ,
6862   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6863   b .value_forbidden:n = true ,
6864   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6865   B .value_forbidden:n = true ,
6866   color .code:n =
6867     \@@_color:n { #1 }
6868     \tl_set_rescan:Nnn
6869     \l_@@_draw_tl

```



```

6870      { \char_set_catcode_other:N ! }
6871      { #1 } ,
6872      color .value_required:n = true ,
6873      respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6874      respect-arraystretch .default:n = true
6875    }

```

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.

```

6876 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

6877 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6878 {

```

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).

```

6879   \peek_remove_spaces:n
6880   {
6881     \tl_if_blank:nTF { #2 }
6882     { \@@_Block_i 1-1 \q_stop }
6883     {
6884       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6885       \@@_Block_i_czech \@@_Block_i
6886       #2 \q_stop
6887     }
6888     { #1 } { #3 } { #4 }
6889   }
6890 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6891 \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`.

```

6892 {
6893   \char_set_catcode_active:N -
6894   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6895 }

```

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.

```

6896 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6897 {

```

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).

```

6898   \bool_lazy_or:nnTF
6899   { \tl_if_blank_p:n { #1 } }
6900   { \str_if_eq_p:nn { #1 } { * } }
6901   { \int_set:Nn \l_tmpa_int { 100 } }
6902   { \int_set:Nn \l_tmpa_int { #1 } }
6903   \bool_lazy_or:nnTF
6904   { \tl_if_blank_p:n { #2 } }
6905   { \str_if_eq_p:nn { #2 } { * } }
6906   { \int_set:Nn \l_tmpb_int { 100 } }
6907   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6908   \int_compare:nNnTF \l_tmpb_int = 1
6909   {
6910     \str_if_empty:NTF \l_@@_hpos_cell_str
6911     { \str_set:Nn \l_@@_hpos_block_str c }
6912     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6913   }
6914   { \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.

```

6915   \keys_set:known:n { NiceMatrix / Block / FirstPass } { #3 }
6916   \tl_set:Nx \l_tmpa_tl
6917   {
6918     { \int_use:N \c@iRow }
6919     { \int_use:N \c@jCol }
6920     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6921     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6922   }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

`{imin}-{jmin}-{imax}-{jmax}`.

If the block is mono-column or mono-row, we have a special treatment. That’s why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```

6923   \bool_if:nTF
6924   {
6925     (
6926       \int_compare_p:nNn { \l_tmpa_int } = 1
6927       ||
6928       \int_compare_p:nNn { \l_tmpb_int } = 1
6929     )
6930     && ! \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.

```

6931     && ! \l_@@_X_bool
6932   }
6933   { \exp_args:Nee \@@_Block_iv:nnnnn }
6934   { \exp_args:Nee \@@_Block_v:nnnnn }
6935   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6936 }

```

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.

```

6937 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6938 {
6939   \int_gincr:N \g_@@_block_box_int
6940   \cs_set_protected_nopar:Npn \diagbox ##1 ##2

```

```

6941 {
6942   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6943   {
6944     \@@_actually_diagbox:nnnnnn
6945     { \int_use:N \c@iRow }
6946     { \int_use:N \c@jCol }
6947     { \int_eval:n { \c@iRow + #1 - 1 } }
6948     { \int_eval:n { \c@jCol + #2 - 1 } }
6949     { \exp_not:n { ##1 } }
6950     { \exp_not:n { ##2 } }
6951   }
6952 }
6953 \box_gclear_new:c
6954 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after, the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

6955 \hbox_gset:cn
6956 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6957 {

```

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}`).

```

6958 \tl_if_empty:NTF \l_@@_color_tl
6959 { \int_compare:nNnT { #2 } = 1 \set@color }
6960 { \@@_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`.

```

6961 \int_compare:nNnT { #1 } = 1
6962 {
6963   \int_if_zero:nTF \c@iRow
6964   \l_@@_code_for_first_row_tl
6965   {
6966     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6967     \l_@@_code_for_last_row_tl
6968   }
6969   \g_@@_row_style_tl
6970 }
6971 \bool_if:NF \l_@@_respect_arraystretch_bool
6972 { \cs_set:Npn \arraystretch { 1 } }
6973 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

6974 #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.).

```

6975 \@@_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}`.

```

6976 \bool_if:NTF \l_@@_tabular_bool
6977 {
6978   \bool_lazy_all:nTF
6979   {
6980     { \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.

```

6981         { \dim_compare:n { \l_@@_col_width_dim >= \c_zero_dim } }
6982         { ! \g_@@_rotate_bool }
6983     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

6984     {
6985         \use:e
6986         {
6987             \exp_not:N \begin { minipage }%
6988             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6989             { \l_@@_col_width_dim }
6990             \str_case:Vn \l_@@_hpos_block_str
6991             { c \centering r \raggedleft l \raggedright }
6992         }
6993         #5
6994     \end { minipage }
6995 }

```

In the other cases, we use a `{tabular}`.

```

6996     {
6997         \use:e
6998         {
6999             \exp_not:N \begin { tabular }%
7000             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7001             { @ { } \l_@@_hpos_block_str @ { } }
7002         }
7003         #5
7004     \end { tabular }
7005 }
7006 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7007     {
7008         \c_math_toggle_token
7009         \use:e
7010         {
7011             \exp_not:N \begin { array }%
7012             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7013             { @ { } \l_@@_hpos_block_str @ { } }
7014         }
7015         #5
7016     \end { array }
7017     \c_math_toggle_token
7018 }
7019 }

```

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).

```

7020     \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.

```

7021     \int_compare:nNnT { #2 } = 1
7022     {
7023         \dim_gset:Nn \g_@@_blocks_wd_dim
7024         {
7025             \dim_max:nn
7026             \g_@@_blocks_wd_dim
7027             {
7028                 \box_wd:c

```

```

7029         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7030     }
7031 }
7032 }

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.

7033 \str_if_eq:VnT \l_@@_vpos_of_block_str { c }
7034 {
7035     \int_compare:nNnT { #1 } = 1
7036     {
7037         \dim_gset:Nn \g_@@_blocks_ht_dim
7038         {
7039             \dim_max:nn
7040             \g_@@_blocks_ht_dim
7041             {
7042                 \box_ht:c
7043                 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7044             }
7045         }
7046         \dim_gset:Nn \g_@@_blocks_dp_dim
7047         {
7048             \dim_max:nn
7049             \g_@@_blocks_dp_dim
7050             {
7051                 \box_dp:c
7052                 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7053             }
7054         }
7055     }
7056 }
7057 \seq_gput_right:Nx \g_@@_blocks_seq
7058 {
7059     \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.

```

7060 {
7061     \exp_not:n { #3 } ,
7062     \l_@@_hpos_block_str ,
7063     \bool_if:NT \g_@@_rotate_bool
7064     {
7065         \bool_if:NTF \g_@@_rotate_c_bool
7066         { v-center }
7067         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7068     }
7069 }
7070 }
7071 {
7072     \box_use_drop:c
7073     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7074 }
7075 }
7076 \bool_set_false:N \g_@@_rotate_c_bool
7077 }

```

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.

```

7078 \cs_new:Npn \@@_adjust_hpos_rotate:
7079 {
7080   \bool_if:NT \g_@@_rotate_bool
7081   {
7082     \str_set:Nx \l_@@_hpos_block_str
7083     {
7084       \bool_if:NTF \g_@@_rotate_c_bool
7085       { c }
7086       {
7087         \str_case:Nf \l_@@_vpos_of_block_str
7088         { b l B l t r T r }
7089         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7090       }
7091     }
7092   }
7093 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7094 \cs_new_protected:Npn \@@_rotate_box_of_block:
7095 {
7096   \box_grotate:cn
7097   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7098   { 90 }
7099   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7100   {
7101     \vbox_gset_top:cn
7102     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7103     {
7104       \skip_vertical:n { 0.8 ex }
7105       \box_use:c
7106       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7107     }
7108   }
7109   \bool_if:NT \g_@@_rotate_c_bool
7110   {
7111     \hbox_gset:cn
7112     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7113     {
7114       \c_math_toggle_token
7115       \vcenter
7116       {
7117         \box_use:c
7118         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7119       }
7120       \c_math_toggle_token
7121     }
7122   }
7123 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column. In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7124 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7125 {
7126   \seq_gput_right:Nx \g_@@_blocks_seq
7127   {
7128     \l_tmpa_tl

```

```

7129 { \exp_not:n { #3 } }
7130 {
7131   \bool_if:NTF \l_@@_tabular_bool
7132   {
7133     \group_begin:
7134     \bool_if:NF \l_@@_respect_arraystretch_bool
7135     { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7136     \exp_not:n
7137     {
7138       \dim_zero:N \extrarowheight
7139       #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).

```

7140       % \@@_adjust_hpos_rotate:
7141       \use:e
7142       {
7143         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
7144         { @ { } \l_@@_hpos_block_str @ { } }
7145       }
7146       #5
7147       \end { tabular }
7148     }
7149     \group_end:
7150   }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7151 {
7152   \group_begin:
7153   \bool_if:NF \l_@@_respect_arraystretch_bool
7154   { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7155   \exp_not:n
7156   {
7157     \dim_zero:N \extrarowheight
7158     #4
7159     % \@@_adjust_hpos_rotate:
7160     \c_math_toggle_token % :n c
7161     \use:e
7162     {
7163       \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
7164       { @ { } \l_@@_hpos_block_str @ { } }
7165     }
7166     #5
7167     \end { array }
7168     \c_math_toggle_token
7169   }
7170   \group_end:
7171 }
7172 }
7173 }
7174 }

```

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).

```

7175 \keys_define:nn { NiceMatrix / Block / SecondPass }
7176 {
7177   tikz .code:n =
7178   \IfPackageLoadedTF { tikz }
7179   { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }

```

```

7180     { \@@_error:n { tikz-key-without-tikz } } ,
7181     tikz .value_required:n = true ,
7182     fill .code:n =
7183       \tl_set_rescan:Nnn
7184       \l_@@_fill_tl
7185       { \char_set_catcode_other:N ! }
7186       { #1 } ,
7187     fill .value_required:n = true ,
7188     opacity .tl_set:N = \l_@@_opacity_tl ,
7189     opacity .value_required:n = true ,
7190     draw .code:n =
7191       \tl_set_rescan:Nnn
7192       \l_@@_draw_tl
7193       { \char_set_catcode_other:N ! }
7194       { #1 } ,
7195     draw .default:n = default ,
7196     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7197     rounded-corners .default:n = 4 pt ,
7198     color .code:n =
7199       \@@_color:n { #1 }
7200       \tl_set_rescan:Nnn
7201       \l_@@_draw_tl
7202       { \char_set_catcode_other:N ! }
7203       { #1 } ,
7204     borders .clist_set:N = \l_@@_borders_clist ,
7205     borders .value_required:n = true ,
7206     hvlines .meta:n = { vlines , hlines } ,
7207     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7208     vlines .default:n = true ,
7209     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7210     hlines .default:n = true ,
7211     line-width .dim_set:N = \l_@@_line_width_dim ,
7212     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7213     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7214     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7215     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7216     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7217       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7218     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7219       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7220     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7221       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7222     t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
7223     T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
7224     b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
7225     B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
7226     v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
7227     v-center .value_forbidden:n = true ,
7228     name .tl_set:N = \l_@@_block_name_str ,
7229     name .value_required:n = true ,
7230     name .initial:n = ,
7231     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
7232     transparent .bool_set:N = \l_@@_transparent_bool ,
7233     transparent .default:n = true ,
7234     transparent .initial:n = false ,
7235     unknown .code:n = \@@_error:n { Unknown-key-for-Block }
7236   }

```

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.


```

7237 \cs_new_protected:Npn \@@_draw_blocks:
7238 {
7239   \cs_set_eq:NN \ialign \@@_old_ialign:
7240   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7241 }
7242 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7243 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7244   \int_zero_new:N \l_@@_last_row_int
7245   \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.

```

7246   \int_compare:nNnTF { #3 } > { 99 }
7247   { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
7248     { \int_set:Nn \l_@@_last_row_int { #3 } }
7249   \int_compare:nNnTF { #4 } > { 99 }
7250   { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
7251     { \int_set:Nn \l_@@_last_col_int { #4 } }
7252   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7253   {
7254     \bool_lazy_and:nnTF
7255       \l_@@_preamble_bool
7256       {
7257         \int_compare_p:n
7258           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7259       }
7260     {
7261       \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
7262       \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
7263       \@@_msg_redirect_name:nn { columns-not-used } { none }
7264     }
7265     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
7266   }
7267   {
7268     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7269     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
7270     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7271   }
7272 }

```

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

```

7273 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7274 {

```

The group is for the keys.

```

7275   \group_begin:
7276   \int_compare:nNnT { #1 } = { #3 }
7277   { \str_set:Nn \l_@@_vpos_of_block_str { t } }
7278   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7279   \bool_if:NT \l_@@_vlines_block_bool
7280   {
7281     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7282     {
7283       \@@_vlines_block:nnn

```

```

7284         { \exp_not:n { #5 } }
7285         { #1 - #2 }
7286         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7287     }
7288 }
7289 \bool_if:NT \l_@@_hlines_block_bool
7290 {
7291     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7292     {
7293         \@@_hlines_block:nnn
7294         { \exp_not:n { #5 } }
7295         { #1 - #2 }
7296         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7297     }
7298 }
7299 \bool_if:nF
7300 {
7301     \l_@@_transparent_bool
7302     || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7303 }
7304 {

```

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).

```

7305     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7306     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7307 }

7308 \bool_lazy_and:nnT
7309 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
7310 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7311 { \@@_error:n { hlines-with-color } }

7312 \tl_if_empty:NF \l_@@_draw_tl
7313 {
7314     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7315     {
7316         \@@_stroke_block:nnn
7317         { \exp_not:n { #5 } } % #5 are the options
7318         { #1 - #2 }
7319         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7320     }
7321     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7322     { { #1 } { #2 } { #3 } { #4 } }
7323 }
7324 \clist_if_empty:NF \l_@@_borders_clist
7325 {
7326     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7327     {
7328         \@@_stroke_borders_block:nnn
7329         { \exp_not:n { #5 } }
7330         { #1 - #2 }
7331         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7332     }
7333 }
7334 \tl_if_empty:NF \l_@@_fill_tl
7335 {
7336     \tl_if_empty:NF \l_@@_opacity_tl
7337     {
7338         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7339             {

```

```

7340         \tl_set:Nx \l_@@_fill_tl
7341         {
7342             [ opacity = \l_@@_opacity_tl ,
7343             \tl_tail:V \l_@@_fill_tl
7344         }
7345     }
7346     {
7347         \tl_set:Nx \l_@@_fill_tl
7348         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7349     }
7350 }
7351 \tl_gput_right:Nx \g_@@_pre_code_before_tl
7352 {
7353     \exp_not:N \roundedrectanglecolor
7354     \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7355         { \l_@@_fill_tl }
7356         { { \l_@@_fill_tl } }
7357         { #1 - #2 }
7358         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7359         { \dim_use:N \l_@@_rounded_corners_dim }
7360     }
7361 }
7362 \seq_if_empty:NF \l_@@_tikz_seq
7363 {
7364     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7365     {
7366         \@@_block_tikz:nnnnn
7367         { #1 }
7368         { #2 }
7369         { \int_use:N \l_@@_last_row_int }
7370         { \int_use:N \l_@@_last_col_int }
7371         { \seq_use:Nn \l_@@_tikz_seq { , } }
7372     }
7373 }
7374 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7375 {
7376     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7377     {
7378         \@@_actually_diagbox:nnnnnn
7379         { #1 }
7380         { #2 }
7381         { \int_use:N \l_@@_last_row_int }
7382         { \int_use:N \l_@@_last_col_int }
7383         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7384     }
7385 }
7386 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7387 \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.

```

7388 \pgfpicture
7389 \pgfrememberpicturepositiononpagetrue
7390 \pgf@relevantforpicturesizefalse
7391 \@@_qpoint:n { row - #1 }
7392 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7393 \@@_qpoint:n { col - #2 }
7394 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7395 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7396 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7397 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7398 \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.

```

7399 \@@_pgf_rect_node:nnnnn
7400 { \@@_env: - #1 - #2 - block }
7401 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7402 \str_if_empty:NF \l_@@_block_name_str
7403 {
7404   \pgfnodealias
7405   { \@@_env: - \l_@@_block_name_str }
7406   { \@@_env: - #1 - #2 - block }
7407   \str_if_empty:NF \l_@@_name_str
7408   {
7409     \pgfnodealias
7410     { \l_@@_name_str - \l_@@_block_name_str }
7411     { \@@_env: - #1 - #2 - block }
7412   }
7413 }

```

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.

```

7414 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7415 {
7416   \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.

```

7417 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7418 {

```

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.

```

7419 \cs_if_exist:cT
7420 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7421 {
7422   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7423   {
7424     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7425     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7426   }
7427 }
7428 }

```

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.

```

7429     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7430     {
7431         \@@_qpoint:n { col - #2 }
7432         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7433     }
7434     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7435     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7436     {
7437         \cs_if_exist:cT
7438         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7439         {
7440             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7441             {
7442                 \pgfpointanchor
7443                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7444                 { east }
7445                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7446             }
7447         }
7448     }
7449     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7450     {
7451         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7452         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7453     }
7454     \@@_pgf_rect_node:nnnnn
7455     { \@@_env: - #1 - #2 - block - short }
7456     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7457 }

```

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.

```

7458     \bool_if:NT \l_@@_medium_nodes_bool
7459     {
7460         \@@_pgf_rect_node:nnn
7461         { \@@_env: - #1 - #2 - block - medium }
7462         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7463         {
7464             \pgfpointanchor
7465             { \@@_env:
7466               - \int_use:N \l_@@_last_row_int
7467               - \int_use:N \l_@@_last_col_int - medium
7468             }
7469             { south-east }
7470         }
7471     }

```

Now, we will put the label of the block.

```

7472     \bool_lazy_any:nTF
7473     {
7474         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7475         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7476         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7477     }
7478     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7479     \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.

```

7480     \bool_if:nT \g_@@_last_col_found_bool
7481     {
7482         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7483         { \str_set:Nn \l_@@_hpos_block_str 1 }
7484     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7485     \tl_set:Nx \l_tmpa_tl
7486     {
7487         \str_case:Vn \l_@@_vpos_of_block_str
7488         {
7489             c {
7490                 \str_case:Vn \l_@@_hpos_block_str
7491                 {
7492                     c { center }
7493                     l { west }
7494                     r { east }
7495                 }
7496             }
7497         }
7498         T {
7499             \str_case:Vn \l_@@_hpos_block_str
7500             {
7501                 c { north }
7502                 l { north-west }
7503                 r { north-east }
7504             }
7505         }
7506         B {
7507             \str_case:Vn \l_@@_hpos_block_str
7508             {
7509                 c { south }
7510                 l { south-west }
7511                 r { south-east }
7512             }
7513         }
7514     }
7515 }
7516 }
7517 }
7518 \pgftransformshift
7519 {
7520     \pgfpointanchor
7521     {
7522         \@@_env: - #1 - #2 - block
7523         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7524     }
7525     { \l_tmpa_tl }
7526 }
7527 \pgfset
7528 {
7529     inner~xsep = \c_zero_dim ,
7530     inner~ysep = \l_@@_block_ysep_dim
7531 }
7532 \pgfnode
7533 { rectangle }
7534 { \l_tmpa_tl }
7535 { \box_use_drop:N \l_@@_cell_box } { } { }
7536 }

```

End of the case when `\l_@@_vpos_of_block_str` is equal to c, T or B. Now, the other cases.

```

7537 {

```

```

7538 \pgfextracty \l_tmpa_dim
7539 {
7540     \@@_qpoint:n
7541     {
7542         row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7543         - base
7544     }
7545 }
7546 \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.

```

7547 \pgfpointanchor
7548 {
7549     \@@_env: - #1 - #2 - block
7550     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7551 }
7552 {
7553     \str_case:Vn \l_@@_hpos_block_str
7554     {
7555         c { center }
7556         l { west }
7557         r { east }
7558     }
7559 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7560 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7561 \pgfset { inner-sep = \c_zero_dim }
7562 \pgfnode
7563 { rectangle }
7564 {
7565     \str_case:Vn \l_@@_hpos_block_str
7566     {
7567         c { base }
7568         l { base-west }
7569         r { base-east }
7570     }
7571 }
7572 { \box_use_drop:N \l_@@_cell_box } { } { }
7573 }
7574 \endpgfpicture
7575 \group_end:
7576 }

```

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).

```

7577 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7578 {
7579     \group_begin:
7580     \tl_clear:N \l_@@_draw_tl
7581     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7582     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7583     \pgfpicture
7584     \pgfrememberpicturepositiononpagetrue
7585     \pgf@relevantforpicturesizefalse
7586     \tl_if_empty:NF \l_@@_draw_tl
7587     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7588     \str_if_eq:VnTF \l_@@_draw_tl { default }
7589     { \CT@arc@ }

```

```

7590     { \@@_color:V \l_@@_draw_tl }
7591   }
7592   \pgfsetcornersarced
7593   {
7594     \pgfpoint
7595     { \l_@@_rounded_corners_dim }
7596     { \l_@@_rounded_corners_dim }
7597   }
7598   \@@_cut_on_hyphen:w #2 \q_stop
7599   \bool_lazy_and:nnT
7600   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7601   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7602   {
7603     \@@_qpoint:n { row - \l_tmpa_tl }
7604     \dim_set_eq:NN \l_tmpb_dim \pgf@y
7605     \@@_qpoint:n { col - \l_tmpb_tl }
7606     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7607     \@@_cut_on_hyphen:w #3 \q_stop
7608     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7609     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7610     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7611     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7612     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7613     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7614     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7615     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7616     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7617     \pgfpathrectanglecorners
7618     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7619     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7620     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7621     { \pgfusepathqstroke }
7622     { \pgfusepath { stroke } }
7623   }
7624   \endpgfpicture
7625   \group_end:
7626 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7627 \keys_define:nn { NiceMatrix / BlockStroke }
7628 {
7629   color .tl_set:N = \l_@@_draw_tl ,
7630   draw .code:n =
7631     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7632   draw .default:n = default ,
7633   line-width .dim_set:N = \l_@@_line_width_dim ,
7634   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7635   rounded-corners .default:n = 4 pt
7636 }

```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

7637 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7638 {
7639   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7640   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7641   \@@_cut_on_hyphen:w #2 \q_stop
7642   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7643   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7644   \@@_cut_on_hyphen:w #3 \q_stop
7645   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7646   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }

```



```

7647 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7648 {
7649   \use:e
7650   {
7651     \@@_vline:n
7652     {
7653       position = ##1 ,
7654       start = \l_@@_tmpc_tl ,
7655       end = \int_eval:n { \l_tmpa_tl - 1 } ,
7656       total-width = \dim_use:N \l_@@_line_width_dim
7657     }
7658   }
7659 }
7660 }
7661 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7662 {
7663   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7664   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7665   \@@_cut_on_hyphen:w #2 \q_stop
7666   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7667   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7668   \@@_cut_on_hyphen:w #3 \q_stop
7669   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7670   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7671   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7672   {
7673     \use:e
7674     {
7675       \@@_hline:n
7676       {
7677         position = ##1 ,
7678         start = \l_@@_tmpd_tl ,
7679         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7680         total-width = \dim_use:N \l_@@_line_width_dim
7681       }
7682     }
7683   }
7684 }

```

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).

```

7685 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7686 {
7687   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7688   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7689   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7690   { \@@_error:n { borders~forbidden } }
7691   {
7692     \tl_clear_new:N \l_@@_borders_tikz_tl
7693     \keys_set:nV
7694       { NiceMatrix / OnlyForTikzInBorders }
7695     \l_@@_borders_clist
7696     \@@_cut_on_hyphen:w #2 \q_stop
7697     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7698     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7699     \@@_cut_on_hyphen:w #3 \q_stop
7700     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7701     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7702     \@@_stroke_borders_block_i:
7703   }
7704 }

```

```

7705 \hook_gput_code:nnn { begindocument } { . }
7706 {
7707   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7708   {
7709     \c_@@_pgfortikzpicture_tl
7710     \@@_stroke_borders_block_ii:
7711     \c_@@_endpgfortikzpicture_tl
7712   }
7713 }
7714 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7715 {
7716   \pgfrememberpicturepositiononpagetrue
7717   \pgf@relevantforpicturesizefalse
7718   \CT@arc@
7719   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7720   \clist_if_in:NnT \l_@@_borders_clist { right }
7721   { \@@_stroke_vertical:n \l_tmpb_tl }
7722   \clist_if_in:NnT \l_@@_borders_clist { left }
7723   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7724   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7725   { \@@_stroke_horizontal:n \l_tmpa_tl }
7726   \clist_if_in:NnT \l_@@_borders_clist { top }
7727   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7728 }
7729 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7730 {
7731   tikz .code:n =
7732     \cs_if_exist:NTF \tikzpicture
7733     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7734     { \@@_error:n { tikz-in~borders~without~tikz } } ,
7735   tikz .value_required:n = true ,
7736   top .code:n = ,
7737   bottom .code:n = ,
7738   left .code:n = ,
7739   right .code:n = ,
7740   unknown .code:n = \@@_error:n { bad~border }
7741 }

```

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).

```

7742 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7743 {
7744   \@@_qpoint:n \l_@@_tmpc_tl
7745   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7746   \@@_qpoint:n \l_tmpa_tl
7747   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7748   \@@_qpoint:n { #1 }
7749   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7750   {
7751     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7752     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7753     \pgfusepathqstroke
7754   }
7755   {
7756     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7757     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7758   }
7759 }

```

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).

```

7760 \cs_new_protected:Npn \@@_stroke_horizontal:n #1

```

```

7761 {
7762   \@@_qpoint:n \l_@@_tmpd_tl
7763   \clist_if_in:NnTF \l_@@_borders_clist { left }
7764     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7765     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7766   \@@_qpoint:n \l_tmpb_tl
7767   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7768   \@@_qpoint:n { #1 }
7769   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7770     {
7771       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7772       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7773       \pgfusepathqstroke
7774     }
7775     {
7776       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7777       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7778     }
7779 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7780 \keys_define:nn { NiceMatrix / BlockBorders }
7781 {
7782   borders .clist_set:N = \l_@@_borders_clist ,
7783   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7784   rounded-corners .default:n = 4 pt ,
7785   line-width .dim_set:N = \l_@@_line_width_dim
7786 }

```

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.

```

7787 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7788 {
7789   \begin { tikzpicture }
7790   \@@_clip_with_rounded_corners:
7791   \clist_map_inline:nn { #5 }
7792     {
7793       \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7794       \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7795       (
7796         [
7797           xshift = \dim_use:N \l_@@_offset_dim ,
7798           yshift = - \dim_use:N \l_@@_offset_dim
7799         ]
7800         #1 -| #2
7801       )
7802       rectangle
7803       (
7804         [
7805           xshift = - \dim_use:N \l_@@_offset_dim ,
7806           yshift = \dim_use:N \l_@@_offset_dim
7807         ]
7808         \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7809       ) ;
7810     }
7811   \end { tikzpicture }
7812 }
7813 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

```

```

7814 \keys_define:nn { NiceMatrix / SpecialOffset }
7815   { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7816 \cs_set_protected:Npn \@@_renew_matrix:
7817 {
7818   \RenewDocumentEnvironment { pmatrix } { } {
7819     { \pNiceMatrix }
7820     { \endpNiceMatrix }
7821   \RenewDocumentEnvironment { vmatrix } { } {
7822     { \vNiceMatrix }
7823     { \endvNiceMatrix }
7824   \RenewDocumentEnvironment { Vmatrix } { } {
7825     { \VNiceMatrix }
7826     { \endVNiceMatrix }
7827   \RenewDocumentEnvironment { bmatrix } { } {
7828     { \bNiceMatrix }
7829     { \endbNiceMatrix }
7830   \RenewDocumentEnvironment { Bmatrix } { } {
7831     { \BNiceMatrix }
7832     { \endBNiceMatrix }
7833 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7834 \keys_define:nn { NiceMatrix / Auto }
7835 {
7836   columns-type .tl_set:N = \l_@@_columns_type_tl ,
7837   columns-type .value_required:n = true ,
7838   l .meta:n = { columns-type = l } ,
7839   r .meta:n = { columns-type = r } ,
7840   c .meta:n = { columns-type = c } ,
7841   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7842   delimiters / color .value_required:n = true ,
7843   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7844   delimiters / max-width .default:n = true ,
7845   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7846   delimiters .value_required:n = true ,
7847   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7848   rounded-corners .default:n = 4 pt
7849 }
7850 \NewDocumentCommand \AutoNiceMatrixWithDelims
7851 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7852 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7853 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7854 {

```

The group is for the protection of the keys.

```

7855   \group_begin:
7856   \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`).

```

7857   % \bool_set_false:N \l_@@_preamble_bool
7858   \use:e
7859   {

```

```

7860 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7861 { * { #4 } { \exp_not:V \l_@@_columns_type_t1 } }
7862 [ \exp_not:V \l_tmpa_t1 ]
7863 }
7864 \int_if_zero:nT \l_@@_first_row_int
7865 {
7866 \int_if_zero:nT \l_@@_first_col_int { & }
7867 \prg_replicate:nn { #4 - 1 } { & }
7868 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7869 }
7870 \prg_replicate:nn { #3 }
7871 {
7872 \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).

```

7873 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7874 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7875 }
7876 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7877 {
7878 \int_if_zero:nT \l_@@_first_col_int { & }
7879 \prg_replicate:nn { #4 - 1 } { & }
7880 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7881 }
7882 \end { NiceArrayWithDelims }
7883 \group_end:
7884 }

7885 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7886 {
7887 \cs_set_protected:cpn { #1 AutoNiceMatrix }
7888 {
7889 \bool_gset_true:N \g_@@_delims_bool
7890 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7891 \AutoNiceMatrixWithDelims { #2 } { #3 }
7892 }
7893 }

7894 \@@_define_com:nnn p ( )
7895 \@@_define_com:nnn b [ ]
7896 \@@_define_com:nnn v | |
7897 \@@_define_com:nnn V \l \l
7898 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

7899 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7900 {
7901 \group_begin:
7902 \bool_gset_false:N \g_@@_delims_bool
7903 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7904 \group_end:
7905 }

```

30 The redefinition of the command \dotfill

```

7906 \cs_set_eq:NN \@@_old_dotfill \dotfill
7907 \cs_new_protected:Npn \@@_dotfill:
7908 {

```

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}`).

```

7909 \@@_old_dotfill
7910 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7911 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

7912 \cs_new_protected:Npn \@@_dotfill_i:
7913 { \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.

```

7914 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7915 {
7916 \tl_gput_right:Nx \g_@@_pre_code_after_tl
7917 {
7918 \@@_actually_diagbox:nnnnnn
7919 { \int_use:N \c@iRow }
7920 { \int_use:N \c@jCol }
7921 { \int_use:N \c@iRow }
7922 { \int_use:N \c@jCol }
7923 { \exp_not:n { #1 } }
7924 { \exp_not:n { #2 } }
7925 }

```

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`.

```

7926 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7927 {
7928 { \int_use:N \c@iRow }
7929 { \int_use:N \c@jCol }
7930 { \int_use:N \c@iRow }
7931 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7932 { }
7933 }
7934 }

```

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.

```

7935 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7936 {
7937 \pgfpicture
7938 \pgf@relevantforpicturesizefalse
7939 \pgfrememberpicturepositiononpagetrue
7940 \@@_qpoint:n { row - #1 }
7941 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7942 \@@_qpoint:n { col - #2 }
7943 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7944 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7945 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7946 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7947 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7948 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

```

7949 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7950 {

```

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.

```

7951 \CT@arc@
7952 \pgfsetroundcap
7953 \pgfusepathqstroke
7954 }
7955 \pgfset { inner~sep = 1 pt }
7956 \pgfscope
7957 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7958 \pgfnode { rectangle } { south~west }
7959 {
7960 \begin { minipage } { 20 cm }
7961 \@@_math_toggle_token: #5 \@@_math_toggle_token:
7962 \end { minipage }
7963 }
7964 { }
7965 { }
7966 \endpgfscope
7967 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7968 \pgfnode { rectangle } { north~east }
7969 {
7970 \begin { minipage } { 20 cm }
7971 \raggedleft
7972 \@@_math_toggle_token: #6 \@@_math_toggle_token:
7973 \end { minipage }
7974 }
7975 { }
7976 { }
7977 \endpgfpicture
7978 }

```

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. 81.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

7979 \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 `\`.

```

7980 \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`.

```

7981 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7982 {
7983 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7984 \@@_CodeAfter_iv:n
7985 }

```

We catch the argument of the command `\end` (in `#1`).

```

7986 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7987 {

```

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.

```

7988   \str_if_eq:eeTF \@currenvir { #1 }
7989   { \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`.

```

7990   {
7991     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7992     \@@_CodeAfter_ii:n
7993   }
7994 }

```

33 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

7995 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7996 {
7997   \pgfpicture
7998   \pgfrememberpicturepositiononpagetrue
7999   \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.

```

8000   \@@_qpoint:n { row - 1 }
8001   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8002   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8003   \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).

```

8004   \bool_if:nTF { #3 }
8005   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8006   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8007   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8008   {
8009     \cs_if_exist:cT
8010     { pgf @ sh @ ns @ \@_env: - ##1 - #2 }
8011     {
8012       \pgfpictureanchor
8013       { \@_env: - ##1 - #2 }
8014       { \bool_if:nTF { #3 } { west } { east } }
8015       \dim_set:Nn \l_tmpa_dim
8016       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8017     }
8018   }

```


Now we can put the delimiter with a node of PGF.

```

8019 \pgfset { inner~sep = \c_zero_dim }
8020 \dim_zero:N \nulldelimiterspace
8021 \pgftransformshift
8022 {
8023   \pgfpoint
8024   { \l_tmpa_dim }
8025   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8026 }
8027 \pgfnode
8028 { rectangle }
8029 { \bool_if:nTF { #3 } { east } { west } }
8030 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8031   \nullfont
8032   \c_math_toggle_token
8033   \@@_color:V \l_@@_delimiters_color_tl
8034   \bool_if:nTF { #3 } { \left #1 } { \left . }
8035   \vcenter
8036   {
8037     \nullfont
8038     \hrule \@height
8039       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8040       \@depth \c_zero_dim
8041       \@width \c_zero_dim
8042   }
8043   \bool_if:nTF { #3 } { \right . } { \right #1 }
8044   \c_math_toggle_token
8045 }
8046 { }
8047 { }
8048 \endpgfpicture
8049 }

```

34 The command \SubMatrix

```

8050 \keys_define:nn { NiceMatrix / sub-matrix }
8051 {
8052   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8053   extra-height .value_required:n = true ,
8054   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8055   left-xshift .value_required:n = true ,
8056   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8057   right-xshift .value_required:n = true ,
8058   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8059   xshift .value_required:n = true ,
8060   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8061   delimiters / color .value_required:n = true ,
8062   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8063   slim .default:n = true ,
8064   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8065   hlines .default:n = all ,
8066   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8067   vlines .default:n = all ,
8068   hvlines .meta:n = { hlines, vlines } ,
8069   hvlines .value_forbidden:n = true
8070 }
8071 \keys_define:nn { NiceMatrix }
8072 {
8073   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,

```

```

8074 NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8075 pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8076 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8077 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8078 \keys_define:nn { NiceMatrix / SubMatrix }
8079 {
8080   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8081   delimiters / color .value_required:n = true ,
8082   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8083   hlines .default:n = all ,
8084   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8085   vlines .default:n = all ,
8086   hvlines .meta:n = { hlines, vlines } ,
8087   hvlines .value_forbidden:n = true ,
8088   name .code:n =
8089     \tl_if_empty:nTF { #1 }
8090     { \@@_error:n { Invalid-name } }
8091     {
8092       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8093       {
8094         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8095         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8096         {
8097           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8098           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8099         }
8100       }
8101       { \@@_error:n { Invalid-name } }
8102     } ,
8103   name .value_required:n = true ,
8104   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8105   rules .value_required:n = true ,
8106   code .tl_set:N = \l_@@_code_tl ,
8107   code .value_required:n = true ,
8108   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8109 }

8110 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8111 {
8112   \peek_remove_spaces:n
8113   {
8114     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8115     {
8116       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8117       [
8118         delimiters / color = \l_@@_delimiters_color_tl ,
8119         hlines = \l_@@_submatrix_hlines_clist ,
8120         vlines = \l_@@_submatrix_vlines_clist ,
8121         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8122         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8123         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8124         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8125         #5
8126       ]
8127     }
8128     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8129   }
8130 }

8131 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8132 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }

```

```

8133 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8134 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8135 {
8136   \seq_gput_right:Nx \g_@@_submatrix_seq
8137   {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8138     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8139     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8140     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8141     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8142   }
8143 }

```

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`.

```

8144 \hook_gput_code:nnn { begindocument } { . }
8145 {
8146   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8147   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8148   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8149   {
8150     \peek_remove_spaces:n
8151     {
8152       \@@_sub_matrix:nnnnnnn
8153       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8154     }
8155   }
8156 }

```

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).

```

8157 \NewDocumentCommand \@@_compute_i_j:nn
8158 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8159 { \@@_compute_i_j:nnnn #1 #2 }
8160 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8161 {
8162   \tl_set:Nn \l_@@_first_i_tl { #1 }
8163   \tl_set:Nn \l_@@_first_j_tl { #2 }
8164   \tl_set:Nn \l_@@_last_i_tl { #3 }
8165   \tl_set:Nn \l_@@_last_j_tl { #4 }
8166   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8167   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8168   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8169   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8170   \tl_if_eq:NnT \l_@@_last_i_tl { last }

```

```

8171 { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8172 \tl_if_eq:NnT \l_@@_last_j_tl { last }
8173 { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8174 }
8175 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8176 {
8177   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8178 \@@_compute_i_j:nn { #2 } { #3 }
8179 % added 6.19b
8180 \int_compare:NnT \l_@@_first_i_tl = \l_@@_last_i_tl
8181 { \cs_set:Npn \arraystretch { 1 } }
8182 \bool_lazy_or:nnTF
8183 { \int_compare_p:Nn \l_@@_last_i_tl > \g_@@_row_total_int }
8184 { \int_compare_p:Nn \l_@@_last_j_tl > \g_@@_col_total_int }
8185 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8186 {
8187   \str_clear_new:N \l_@@_submatrix_name_str
8188   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8189   \pgfpicture
8190   \pgfrememberpicturepositiononpagetrue
8191   \pgf@relevantforpicturesizefalse
8192   \pgfset { inner-sep = \c_zero_dim }
8193   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8194   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currfication.

```

8195 \bool_if:NTF \l_@@_submatrix_slim_bool
8196 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8197 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8198 {
8199   \cs_if_exist:cT
8200   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8201   {
8202     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8203     \dim_set:Nn \l_@@_x_initial_dim
8204     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8205   }
8206   \cs_if_exist:cT
8207   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8208   {
8209     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8210     \dim_set:Nn \l_@@_x_final_dim
8211     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8212   }
8213 }
8214 \dim_compare:NnTF \l_@@_x_initial_dim = \c_max_dim
8215 { \@@_error:nn { Impossible-delimiter } { left } }
8216 {
8217   \dim_compare:NnTF \l_@@_x_final_dim = { - \c_max_dim }
8218   { \@@_error:nn { Impossible-delimiter } { right } }
8219   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8220 }
8221 \endpgfpicture
8222 }
8223 \group_end:
8224 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8225 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8226 {
8227   \@@_qpoint:n { row - \l_@@_first_i_tl - base }

```

```

8228 \dim_set:Nn \l_@@_y_initial_dim
8229 {
8230   \fp_to_dim:n
8231   {
8232     \pgf@y
8233     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8234   }
8235   } % modified 6.13c
8236 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8237 \dim_set:Nn \l_@@_y_final_dim
8238 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8239 % modified 6.13c
8240 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8241 {
8242   \cs_if_exist:cT
8243   { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8244   {
8245     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8246     \dim_set:Nn \l_@@_y_initial_dim
8247     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8248   }
8249   \cs_if_exist:cT
8250   { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8251   {
8252     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8253     \dim_set:Nn \l_@@_y_final_dim
8254     { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8255   }
8256 }
8257 \dim_set:Nn \l_tmpa_dim
8258 {
8259   \l_@@_y_initial_dim - \l_@@_y_final_dim +
8260   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8261 }
8262 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8263 \group_begin:
8264 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8265 \@@_set_CT@arc@:V \l_@@_rules_color_tl
8266 \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`.

```

8267 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8268 {
8269   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8270   {
8271     \int_compare:nNnT
8272     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8273     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8274     \@@_qpoint:n { col - ##1 }
8275     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8276     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8277     \pgfusepathqstroke
8278   }
8279 }
8280 }

```

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.

```

8281 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
8282 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8283 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8284 {
8285   \bool_lazy_and:nnTF
8286   { \int_compare_p:nNn { ##1 } > 0 }
8287   {
8288     \int_compare_p:nNn
8289     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8290   {
8291     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8292     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8293     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8294     \pgfusepathqstroke
8295   }
8296   { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
8297 }

```

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.

```

8298 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
8299 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8300 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8301 {
8302   \bool_lazy_and:nnTF
8303   { \int_compare_p:nNn { ##1 } > 0 }
8304   {
8305     \int_compare_p:nNn
8306     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8307   {
8308     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8309 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8310 \dim_set:Nn \l_tmpa_dim
8311 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8312 \str_case:nn { #1 }
8313 {
8314   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8315   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8316   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8317   }
8318   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8319 \dim_set:Nn \l_tmpb_dim
8320 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8321 \str_case:nn { #2 }
8322 {
8323   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8324   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8325   \{ { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8326   }
8327   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8328   \pgfusepathqstroke
8329   \group_end:
8330 }
8331 { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8332 }

```

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).

```

8333 \str_if_empty:NF \l_@@_submatrix_name_str
8334 {
8335     \@@_pgf_rect_node:nnnn \l_@@_submatrix_name_str
8336     \l_@@_x_initial_dim \l_@@_y_initial_dim
8337     \l_@@_x_final_dim \l_@@_y_final_dim
8338 }
8339 \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`.

```

8340 \begin { pgfscope }
8341 \pgftransformshift
8342 {
8343     \pgfpoint
8344     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8345     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8346 }
8347 \str_if_empty:NTF \l_@@_submatrix_name_str
8348 { \@@_node_left:nn #1 { } }
8349 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8350 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8351 \pgftransformshift
8352 {
8353     \pgfpoint
8354     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8355     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8356 }
8357 \str_if_empty:NTF \l_@@_submatrix_name_str
8358 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8359 {
8360     \@@_node_right:nnnn #2
8361     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8362 }
8363 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8364 \flag_clear_new:n { nicematrix }
8365 \l_@@_code_tl
8366 }

```

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`.

```

8367 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument `#1` of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

8368 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8369 {
8370     \use:e
8371     { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8372 }

```

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 -.

```
8373 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8374 { #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.

```
8375 \tl_const:Nn \c_@@_integers_alist_tl
8376 {
8377 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8378 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8379 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8380 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8381 }
```

```
8382 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8383 {
```

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`.

```
8384 \tl_if_empty:nTF { #2 }
8385 {
8386 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8387 {
8388 \flag_raise:n { nicematrix }
8389 \int_if_even:nTF { \flag_height:n { nicematrix } }
8390 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8391 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8392 }
8393 { #1 }
8394 }
```

If there is an hyphen, we have to see whether we have a node of the form i - j , row- i or col- j .

```
8395 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8396 }
```

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`).

```
8397 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8398 {
8399 \str_case:nnF { #1 }
8400 {
8401 { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8402 { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8403 }
```

Now the case of a node of the form i - j .

```
8404 {
8405 \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8406 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8407 }
8408 }
```


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`).

```

8409 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8410 {
8411   \pgfnode
8412     { rectangle }
8413     { east }
8414     {
8415       \nullfont
8416       \c_math_toggle_token
8417       \@@_color:V \l_@@_delimiters_color_tl
8418       \left #1
8419       \vcenter
8420         {
8421           \nullfont
8422           \hrule \@height \l_tmpa_dim
8423             \@depth \c_zero_dim
8424             \@width \c_zero_dim
8425         }
8426       \right .
8427       \c_math_toggle_token
8428     }
8429     { #2 }
8430     { }
8431 }

```

The command `\@@_node_right:nnn` 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.

```

8432 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8433 {
8434   \pgfnode
8435     { rectangle }
8436     { west }
8437     {
8438       \nullfont
8439       \c_math_toggle_token
8440       \@@_color:V \l_@@_delimiters_color_tl
8441       \left .
8442       \vcenter
8443         {
8444           \nullfont
8445           \hrule \@height \l_tmpa_dim
8446             \@depth \c_zero_dim
8447             \@width \c_zero_dim
8448         }
8449       \right #1
8450       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8451       ^ { \smash { #4 } }
8452       \c_math_toggle_token
8453     }
8454     { #2 }
8455     { }
8456 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8457 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8458 {
8459   \peek_remove_spaces:n
8460   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8461 }
8462 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8463 {
8464   \peek_remove_spaces:n
8465   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8466 }
8467 \keys_define:nn { NiceMatrix / Brace }
8468 {
8469   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8470   left-shorten .default:n = true ,
8471   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8472   shorten .meta:n = { left-shorten , right-shorten } ,
8473   right-shorten .default:n = true ,
8474   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8475   yshift .value_required:n = true ,
8476   yshift .initial:n = \c_zero_dim ,
8477   color .tl_set:N = \l_tmpa_tl ,
8478   color .value_required:n = true ,
8479   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8480 }

```

#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*.

```

8481 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8482 {
8483   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8484 \@@_compute_i_j:nn { #1 } { #2 }
8485 \bool_lazy_or:nnTF
8486 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8487 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8488 {
8489   \str_if_eq:nnTF { #5 } { under }
8490   { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8491   { \@@_error:nn { Construct~too~large } { \OverBrace } }
8492 }
8493 {
8494   \tl_clear:N \l_tmpa_tl
8495   \keys_set:nn { NiceMatrix / Brace } { #4 }
8496   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8497   \pgfpicture
8498   \pgfrememberpicturepositiononpagetrue
8499   \pgf@relevantforpicturesizefalse
8500   \bool_if:NT \l_@@_brace_left_shorten_bool
8501   {
8502     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8503     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8504     {
8505       \cs_if_exist:cT
8506       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8507       {
8508         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8509         \dim_set:Nn \l_@@_x_initial_dim
8510         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8511       }
8512     }

```

```

8513     }
8514     \bool_lazy_or:nnT
8515     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8516     { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8517     {
8518         \@@_qpoint:n { col - \l_@@_first_j_tl }
8519         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8520     }
8521     \bool_if:NT \l_@@_brace_right_shorten_bool
8522     {
8523         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8524         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8525         {
8526             \cs_if_exist:cT
8527             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8528             {
8529                 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8530                 \dim_set:Nn \l_@@_x_final_dim
8531                 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8532             }
8533         }
8534     }
8535     \bool_lazy_or:nnT
8536     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8537     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8538     {
8539         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8540         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8541     }
8542     \pgfset { inner-sep = \c_zero_dim }
8543     \str_if_eq:nnTF { #5 } { under }
8544     { \@@_underbrace_i:n { #3 } }
8545     { \@@_overbrace_i:n { #3 } }
8546     \endpgfpicture
8547 }
8548 \group_end:
8549 }

```

The argument is the text to put above the brace.

```

8550 \cs_new_protected:Npn \@@_overbrace_i:n #1
8551 {
8552     \@@_qpoint:n { row - \l_@@_first_i_tl }
8553     \pgftransformshift
8554     {
8555         \pgfpoint
8556         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8557         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8558     }
8559     \pgfnode
8560     { rectangle }
8561     { south }
8562     {
8563         \vbox_top:n
8564         {
8565             \group_begin:
8566             \everycr { }
8567             \halign
8568             {
8569                 \hfil ## \hfil \crcr
8570                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8571                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8572                 \c_math_toggle_token
8573                 \overbrace
8574                 {

```

```

8575             \hbox_to_wd:nn
8576             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8577             { }
8578         }
8579         \c_math_toggle_token
8580         \cr
8581     }
8582     \group_end:
8583 }
8584 }
8585 { }
8586 { }
8587 }

```

The argument is the text to put under the brace.

```

8588 \cs_new_protected:Npn \@@_underbrace_i:n #1
8589 {
8590     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8591     \pgftransformshift
8592     {
8593         \pgfpoint
8594         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8595         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8596     }
8597     \pgfnode
8598     { rectangle }
8599     { north }
8600     {
8601         \group_begin:
8602         \everycr { }
8603         \vbox:n
8604         {
8605             \halign
8606             {
8607                 \hfil ## \hfil \crrc
8608                 \c_math_toggle_token
8609                 \underbrace
8610                 {
8611                     \hbox_to_wd:nn
8612                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8613                     { }
8614                 }
8615                 \c_math_toggle_token
8616                 \cr
8617                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8618                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8619             }
8620         }
8621         \group_end:
8622     }
8623     { }
8624     { }
8625 }

```

36 The command TikzEveryCell

```

8626 \bool_new:N \l_@@_not_empty_bool
8627 \bool_new:N \l_@@_empty_bool
8628

```

```

8629 \keys_define:nn { NiceMatrix / TikzEveryCell }
8630 {
8631   not-empty .code:n =
8632     \bool_lazy_or:nnTF
8633       \l_@@_in_code_after_bool
8634       \g_@@_recreate_cell_nodes_bool
8635       { \bool_set_true:N \l_@@_not_empty_bool }
8636       { \@@_error:n { detection-of~empty-cells } } ,
8637   not-empty .value_forbidden:n = true ,
8638   empty .code:n =
8639     \bool_lazy_or:nnTF
8640       \l_@@_in_code_after_bool
8641       \g_@@_recreate_cell_nodes_bool
8642       { \bool_set_true:N \l_@@_empty_bool }
8643       { \@@_error:n { detection-of~empty-cells } } ,
8644   empty .value_forbidden:n = true ,
8645   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8646 }
8647
8648
8649 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8650 {
8651   \IfPackageLoadedTF { tikz }
8652   {
8653     \group_begin:
8654     \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.

```

8655     \tl_set:Nn \l_tmpa_tl { { #2 } }
8656     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8657       { \@@_for_a_block:nnnnn ##1 }
8658     \@@_all_the_cells:
8659     \group_end:
8660   }
8661   { \@@_error:n { TikzEveryCell~without~tikz } }
8662 }
8663
8664 \tl_new:N \@@_i_tl
8665 \tl_new:N \@@_j_tl
8666
8667 \cs_new_protected:Nn \@@_all_the_cells:
8668 {
8669   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8670   {
8671     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8672     {
8673       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8674       {
8675         \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8676           { \@@_i_tl - \@@_j_tl }
8677         {
8678           \bool_set_false:N \l_tmpa_bool
8679           \cs_if_exist:cTF
8680             { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8681             {
8682               \bool_if:NF \l_@@_empty_bool
8683               { \bool_set_true:N \l_tmpa_bool }
8684             }
8685             {
8686               \bool_if:NF \l_@@_not_empty_bool
8687               { \bool_set_true:N \l_tmpa_bool }
8688             }
8689           \bool_if:NT \l_tmpa_bool

```

```

8690         {
8691             \@@_block_tikz:nnnnV
8692             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8693         }
8694     }
8695 }
8696 }
8697 }
8698 }
8699
8700 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8701 {
8702     \bool_if:NF \l_@@_empty_bool
8703     {
8704         \@@_block_tikz:nnnnV
8705         { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8706     }
8707     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8708 }
8709
8710 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8711 {
8712     \int_step_inline:nnn { #1 } { #3 }
8713     {
8714         \int_step_inline:nnn { #2 } { #4 }
8715         { \cs_set:cpn { cell - ##1 - ####1 } { } }
8716     }
8717 }

```

37 The command \ShowCellNames

```

8718 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8719 {
8720     \dim_zero_new:N \g_@@_tmpc_dim
8721     \dim_zero_new:N \g_@@_tmpd_dim
8722     \dim_zero_new:N \g_@@_tmpe_dim
8723     \int_step_inline:nn \c@iRow
8724     {
8725         \begin { pgfpicture }
8726         \@@_qpoint:n { row - ##1 }
8727         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8728         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8729         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8730         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8731         \bool_if:NTF \l_@@_in_code_after_bool
8732         \end { pgfpicture }
8733         \int_step_inline:nn \c@jCol
8734         {
8735             \hbox_set:Nn \l_tmpa_box
8736             { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
8737             \begin { pgfpicture }
8738             \@@_qpoint:n { col - ####1 }
8739             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8740             \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8741             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8742             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8743             \endpgfpicture
8744             \end { pgfpicture }
8745             \fp_set:Nn \l_tmpa_fp
8746             {
8747                 \fp_min:nn
8748                 {

```

```

8749         \fp_min:nn
8750         { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8751         { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8752     }
8753     { 1.0 }
8754 }
8755 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8756 \pgfpicture
8757 \pgfrememberpicturepositiononpagetrue
8758 \pgf@relevantforpicturesizefalse
8759 \pgftransformshift
8760 {
8761     \pgfpoint
8762     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8763     { \dim_use:N \g_tmpa_dim }
8764 }
8765 \pgfnode
8766 { rectangle }
8767 { center }
8768 { \box_use:N \l_tmpa_box }
8769 { }
8770 { }
8771 \endpgfpicture
8772 }
8773 }
8774 }
8775 \NewDocumentCommand \@@_ShowCellNames { }
8776 {
8777     \bool_if:NT \l_@@_in_code_after_bool
8778     {
8779         \pgfpicture
8780         \pgfrememberpicturepositiononpagetrue
8781         \pgf@relevantforpicturesizefalse
8782         \pgfpathrectanglecorners
8783         { \@@_qpoint:n { 1 } }
8784         {
8785             \@@_qpoint:n
8786             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8787         }
8788         \pgfsetfillopacity { 0.75 }
8789         \pgfsetfillcolor { white }
8790         \pgfusepathqfill
8791         \endpgfpicture
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         \bool_if:NTF \l_@@_in_code_after_bool
8799         {
8800             \pgfpicture
8801             \pgfrememberpicturepositiononpagetrue
8802             \pgf@relevantforpicturesizefalse
8803         }
8804         { \begin { pgfpicture } }
8805         \@@_qpoint:n { row - ##1 }
8806         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8807         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8808         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8809         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8810         \bool_if:NTF \l_@@_in_code_after_bool
8811         { \endpgfpicture }

```

```

8812 { \end { pgfpicture } }
8813 \int_step_inline:nn \c@jCol
8814 {
8815   \hbox_set:Nn \l_tmpa_box
8816   {
8817     \normalfont \Large \sffamily \bfseries
8818     \bool_if:NTF \l_@@_in_code_after_bool
8819     { \color { red } }
8820     { \color { red ! 50 } }
8821     ##1 - ####1
8822   }
8823   \bool_if:NTF \l_@@_in_code_after_bool
8824   {
8825     \pgfpicture
8826     \pgfrememberpicturepositiononpagetrue
8827     \pgf@relevantforpicturesizefalse
8828   }
8829   { \begin { pgfpicture } }
8830   \@@_qpoint:n { col - ####1 }
8831   \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8832   \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8833   \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8834   \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8835   \bool_if:NTF \l_@@_in_code_after_bool
8836   { \endpgfpicture }
8837   { \end { pgfpicture } }
8838   \fp_set:Nn \l_tmpa_fp
8839   {
8840     \fp_min:nn
8841     {
8842       \fp_min:nn
8843       { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8844       { \dim_ratio:nn { \g_@@_tmpe_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8845     }
8846     { 1.0 }
8847   }
8848   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8849   \pgfpicture
8850   \pgfrememberpicturepositiononpagetrue
8851   \pgf@relevantforpicturesizefalse
8852   \pgftransformshift
8853   {
8854     \pgfpoint
8855     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8856     { \dim_use:N \g_@@_tmpa_dim }
8857   }
8858   \pgfnode
8859   { rectangle }
8860   { center }
8861   { \box_use:N \l_tmpa_box }
8862   { }
8863   { }
8864   \endpgfpicture
8865 }
8866 }
8867 }

```

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.

```
8868 \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.

```
8869 \bool_new:N \g_@@_footnote_bool
```

```
8870 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
```

```
8871 {
8872   The-key~'\l_keys_key_str'~is-unknown. \\
8873   That-key-will-be-ignored. \\
8874   For-a-list-of-the-available-keys,~type-H~<return>.
8875 }
```

```
8876 {
8877   The-available-keys-are~(in~alphabetic~order):~
8878   footnote,~
8879   footnotehyper,~
8880   messages-for-Overleaf,~
8881   no-test-for-array,~
8882   renew-dots,~and~
8883   renew-matrix.
8884 }
```

```
8885 \keys_define:nn { NiceMatrix / Package }
```

```
8886 {
8887   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8888   renew-dots .value_forbidden:n = true ,
8889   renew-matrix .code:n = \@@_renew_matrix: ,
8890   renew-matrix .value_forbidden:n = true ,
8891   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
8892   footnote .bool_set:N = \g_@@_footnote_bool ,
8893   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
8894   no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
8895   no-test-for-array .default:n = true ,
8896   unknown .code:n = \@@_error:n { Unknown-key-for-package }
8897 }
```

```
8898 \ProcessKeysOptions { NiceMatrix / Package }
```

```
8899 \@@_msg_new:nn { footnote-with-footnotehyper-package }
```

```
8900 {
8901   You-can't-use-the-option-'footnote'~because-the-package~
8902   footnotehyper~has~already~been~loaded.~
8903   If-you-want,~you-can-use-the-option-'footnotehyper'~and~the~footnotes~
8904   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8905   of~the~package~footnotehyper.\\
8906   The~package~footnote~won't~be~loaded.
8907 }
```

```
8908 \@@_msg_new:nn { footnotehyper-with-footnote-package }
```

```
8909 {
8910   You-can't-use-the-option-'footnotehyper'~because-the-package~
8911   footnote~has~already~been~loaded.~
8912   If-you-want,~you-can-use-the-option-'footnote'~and~the~footnotes~
8913   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8914   of~the~package~footnote.\\
8915   The~package~footnotehyper~won't~be~loaded.
8916 }
```

```
8917 \bool_if:NT \g_@@_footnote_bool
```

```
8918 {
```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8919 \IfClassLoadedTF { beamer }
8920 { \bool_set_false:N \g_@@_footnote_bool }
8921 {
8922   \IfPackageLoadedTF { footnotehyper }
8923   { \@@_error:n { footnote~with~footnotehyper~package } }
8924   { \usepackage { footnote } }
8925 }
8926 }

8927 \bool_if:NT \g_@@_footnotehyper_bool
8928 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8929 \IfClassLoadedTF { beamer }
8930 { \bool_set_false:N \g_@@_footnote_bool }
8931 {
8932   \IfPackageLoadedTF { footnote }
8933   { \@@_error:n { footnotehyper~with~footnote~package } }
8934   { \usepackage { footnotehyper } }
8935 }
8936 \bool_set_true:N \g_@@_footnote_bool
8937 }

```

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.

```

8938 \bool_new:N \l_@@_underscore_loaded_bool
8939 \IfPackageLoadedTF { underscore }
8940 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8941 { }

8942 \hook_gput_code:nnn { begindocument } { . }
8943 {
8944   \bool_if:NF \l_@@_underscore_loaded_bool
8945   {
8946     \IfPackageLoadedTF { underscore }
8947     { \@@_error:n { underscore~after~nicematrix } }
8948     { }
8949   }
8950 }

```

40 Error messages of the package

```

8951 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
8952 { \str_const:Nn \c_@@_available_keys_str { } }
8953 {
8954   \str_const:Nn \c_@@_available_keys_str
8955   { For~a~list~of~the~available~keys,~type~H~<return>. }
8956 }

```

```

8957 \seq_new:N \g_@@_types_of_matrix_seq
8958 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8959 {
8960   NiceMatrix ,
8961   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8962 }
8963 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8964 { \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`.

```

8965 \cs_new_protected:Npn \@@_error_too_much_cols:
8966 {
8967   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8968   {
8969     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8970     { \@@_fatal:n { too-much-cols-for-matrix } }
8971     {
8972       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8973       { \@@_fatal:n { too-much-cols-for-matrix } }
8974       {
8975         \bool_if:NF \l_@@_last_col_without_value_bool
8976         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
8977       }
8978     }
8979   }
8980   { \@@_fatal:nn { too-much-cols-for-array } }
8981 }

```

The following command must *not* be protected since it's used in an error message.

```

8982 \cs_new:Npn \@@_message_hdotsfor:
8983 {
8984   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8985   { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
8986 }
8987 \@@_msg_new:nn { hvlines,~rounded-corners-and~corners }
8988 {
8989   Incompatible~options.\\
8990   You-should-not-use-'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
8991   The~output~will~not~be~reliable.
8992 }
8993 \@@_msg_new:nn { negative-weight }
8994 {
8995   Negative~weight.\\
8996   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
8997   the~value~'\int_use:N \l_@@_weight_int'.\\
8998   The~absolute~value~will~be~used.
8999 }
9000 \@@_msg_new:nn { last-col-not-used }
9001 {
9002   Column~not~used.\\
9003   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9004   in~your~\@@_full_name_env:~.~However,~you~can~go~on.
9005 }
9006 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9007 {
9008   Too~much~columns.\\
9009   In~the~row~\int_eval:n { \c@iRow },~
9010   you~try~to~use~more~columns~
9011   than~allowed~by~your~\@@_full_name_env:~.\@@_message_hdotsfor:\

```

```

9012 The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9013 (plus~the~exterior~columns).~This~error~is~fatal.
9014 }
9015 \@@_msg_new:nn { too~much~cols~for~matrix }
9016 {
9017 Too~much~columns.\\
9018 In~the~row~\int_eval:n { \c@iRow },~
9019 you~try~to~use~more~columns~than~allowed~by~your~
9020 \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
9021 number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9022 columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9023 Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9024 \token_to_str:N \setcounter\ to~change~that~value).~
9025 This~error~is~fatal.
9026 }
9027 \@@_msg_new:nn { too~much~cols~for~array }
9028 {
9029 Too~much~columns.\\
9030 In~the~row~\int_eval:n { \c@iRow },~
9031 ~you~try~to~use~more~columns~than~allowed~by~your~
9032 \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9033 \int_use:N \g_@@_static_num_of_col_int\
9034 ~(plus~the~potential~exterior~ones).
9035 This~error~is~fatal.
9036 }
9037 \@@_msg_new:nn { columns~not~used }
9038 {
9039 Columns~not~used.\\
9040 The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9041 \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9042 The~columns~you~did~not~used~won't~be~created.\\
9043 You~won't~have~similar~error~till~the~end~of~the~document.
9044 }
9045 \@@_msg_new:nn { in~first~col }
9046 {
9047 Erroneous~use.\\
9048 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9049 That~command~will~be~ignored.
9050 }
9051 \@@_msg_new:nn { in~last~col }
9052 {
9053 Erroneous~use.\\
9054 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9055 That~command~will~be~ignored.
9056 }
9057 \@@_msg_new:nn { in~first~row }
9058 {
9059 Erroneous~use.\\
9060 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9061 That~command~will~be~ignored.
9062 }
9063 \@@_msg_new:nn { in~last~row }
9064 {
9065 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9066 That~command~will~be~ignored.
9067 }
9068 \@@_msg_new:nn { caption~outside~float }
9069 {
9070 Key~caption~forbidden.\\
9071 You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~

```

```

9072     environment.~This~key~will~be~ignored.
9073 }

9074 \@@_msg_new:nn { short-caption~without~caption }
9075 {
9076     You~should~not~use~the~key~'short-caption'~without~'caption'.~
9077     However,~your~'short-caption'~will~be~used~as~'caption'.
9078 }

9079 \@@_msg_new:nn { double-closing-delimiter }
9080 {
9081     Double~delimiter.\\
9082     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9083     delimiter.~This~delimiter~will~be~ignored.
9084 }

9085 \@@_msg_new:nn { delimiter~after~opening }
9086 {
9087     Double~delimiter.\\
9088     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9089     delimiter.~That~delimiter~will~be~ignored.
9090 }

9091 \@@_msg_new:nn { bad-option~for~line-style }
9092 {
9093     Bad~line~style.\\
9094     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9095     is~'standard'.~That~key~will~be~ignored.
9096 }

9097 \@@_msg_new:nn { Identical~notes~in~caption }
9098 {
9099     Identical~tabular~notes.\\
9100     You~can't~put~several~notes~with~the~same~content~in~
9101     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9102     If~you~go~on,~the~output~will~probably~be~erroneous.
9103 }

9104 \@@_msg_new:nn { tabularnote~below~the~tabular }
9105 {
9106     \token_to_str:N \tabularnote\ forbidden\\
9107     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9108     of~your~tabular~because~the~caption~will~be~composed~below~
9109     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9110     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9111     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9112     no~similar~error~will~raised~in~this~document.
9113 }

9114 \@@_msg_new:nn { Unknown~key~for~rules }
9115 {
9116     Unknown~key.\\
9117     There~is~only~two~keys~available~here:~width~and~color.\\
9118     Your~key~'\l_keys_key_str'~will~be~ignored.
9119 }

9120 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9121 {
9122     Unknown~key.\\
9123     There~is~only~two~keys~available~here:~
9124     'empty'~and~'not-empty'.\\
9125     Your~key~'\l_keys_key_str'~will~be~ignored.
9126 }

9127 \@@_msg_new:nn { Unknown~key~for~rotate }
9128 {
9129     Unknown~key.\\
9130     The~only~key~available~here~is~'c'.\\
9131     Your~key~'\l_keys_key_str'~will~be~ignored.

```

```

9132 }
9133 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9134 {
9135   Unknown~key.\
9136   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9137   It~you~go~on,~you~will~probably~have~other~errors. \
9138   \c_@@_available_keys_str
9139 }
9140 {
9141   The~available~keys~are~(in~alphabetic~order):~
9142   ccommand,~
9143   color,~
9144   command,~
9145   dotted,~
9146   letter,~
9147   multiplicity,~
9148   sep-color,~
9149   tikz,~and~total~width.
9150 }
9151 \@@_msg_new:nnn { Unknown~key~for~xdots }
9152 {
9153   Unknown~key.\
9154   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\
9155   \c_@@_available_keys_str
9156 }
9157 {
9158   The~available~keys~are~(in~alphabetic~order):~
9159   'color',~
9160   'horizontal~labels',~
9161   'inter',~
9162   'line~style',~
9163   'radius',~
9164   'shorten',~
9165   'shorten~end'~and~'shorten~start'.
9166 }
9167 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9168 {
9169   Unknown~key.\
9170   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
9171   (and~you~try~to~use~'\l_keys_key_str')\
9172   That~key~will~be~ignored.
9173 }
9174 \@@_msg_new:nn { label~without~caption }
9175 {
9176   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9177   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9178 }
9179 \@@_msg_new:nn { W~warning }
9180 {
9181   Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9182   (row~\int_use:N \c@iRow).
9183 }
9184 \@@_msg_new:nn { Construct~too~large }
9185 {
9186   Construct~too~large.\
9187   Your~command~\token_to_str:N #1
9188   can't~be~drawn~because~your~matrix~is~too~small.\
9189   That~command~will~be~ignored.
9190 }
9191 \@@_msg_new:nn { underscore~after~nicematrix }
9192 {

```

```

9193 Problem~with~'underscore'.\\
9194 The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9195 You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9196 '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}}'.
9197 }
9198 \@@_msg_new:nn { ampersand~in~light-syntax }
9199 {
9200 Ampersand~forbidden.\\
9201 You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9202 ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9203 }
9204 \@@_msg_new:nn { double-backslash~in~light-syntax }
9205 {
9206 Double~backslash~forbidden.\\
9207 You~can't~use~\token_to_str:N
9208 \\~to~separate~rows~because~the~key~'light-syntax'~
9209 is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9210 (set~by~the~key~'end-of-row').~This~error~is~fatal.
9211 }
9212 \@@_msg_new:nn { hlines~with~color }
9213 {
9214 Incompatible~keys.\\
9215 You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9216 '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9217 Maybe~it~will~possible~in~future~version.\\
9218 Your~key~will~be~discarded.
9219 }
9220 \@@_msg_new:nn { bad~value~for~baseline }
9221 {
9222 Bad~value~for~baseline.\\
9223 The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9224 valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9225 \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9226 the~form~'line-i'.\\
9227 A~value~of~1~will~be~used.
9228 }
9229 \@@_msg_new:nn { detection~of~empty~cells }
9230 {
9231 Problem~with~'not-empty'\\
9232 For~technical~reasons,~you~must~activate~
9233 'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9234 in~order~to~use~the~key~'\l_keys_key_str'.\\
9235 That~key~will~be~ignored.
9236 }
9237 \@@_msg_new:nn { siunitx~not~loaded }
9238 {
9239 siunitx~not~loaded\\
9240 You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9241 That~error~is~fatal.
9242 }
9243 \@@_msg_new:nn { ragged2e~not~loaded }
9244 {
9245 You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9246 your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9247 \l_keys_key_str'~will~be~used~instead.
9248 }
9249 \@@_msg_new:nn { Invalid~name }
9250 {
9251 Invalid~name.\\
9252 You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N

```

```

9253 \SubMatrix\ of~your~\@@_full_name_env:.\
9254 A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
9255 This~key~will~be~ignored.
9256 }

9257 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9258 {
9259   Wrong~line.\
9260   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9261   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9262   number~is~not~valid.~It~will~be~ignored.
9263 }

9264 \@@_msg_new:nn { Impossible~delimiter }
9265 {
9266   Impossible~delimiter.\
9267   It's~impossible~to~draw~the~#1~delimiter~of~your~
9268   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9269   in~that~column.
9270   \bool_if:NT \l_@@_submatrix_slim_bool
9271     { ~Maybe~you~should~try~without~the~key~'slim'. } \
9272   This~\token_to_str:N \SubMatrix\ will~be~ignored.
9273 }

9274 \@@_msg_new:nnn { width~without~X~columns }
9275 {
9276   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9277   That~key~will~be~ignored.
9278 }
9279 {
9280   This~message~is~the~message~'width~without~X~columns'~
9281   of~the~module~'nicematrix'.~
9282   The~experimented~users~can~disable~that~message~with~
9283   \token_to_str:N \msg_redirect_name:nnn.\
9284 }
9285

9286 \@@_msg_new:nn { key~multiplicity~with~dotted }
9287 {
9288   Incompatible~keys. \
9289   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9290   in~a~'custom~line'.~They~are~incompatible. \
9291   The~key~'multiplicity'~will~be~discarded.
9292 }

9293 \@@_msg_new:nn { empty~environment }
9294 {
9295   Empty~environment.\
9296   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9297 }

9298 \@@_msg_new:nn { No~letter~and~no~command }
9299 {
9300   Erroneous~use.\
9301   Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
9302   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9303   '~'ccommand'~(to~draw~horizontal~rules).\
9304   However,~you~can~go~on.
9305 }

9306 \@@_msg_new:nn { Forbidden~letter }
9307 {
9308   Forbidden~letter.\
9309   You~can't~use~the~letter~'#1'~for~a~customized~line.\
9310   It~will~be~ignored.
9311 }

9312 \@@_msg_new:nn { Several~letters }

```



```

9313 {
9314     Wrong-name.\
9315     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9316     have~used~'\l_@@_letter_str').\
9317     It~will~be~ignored.
9318 }
9319 \@@_msg_new:nn { Delimiter~with~small }
9320 {
9321     Delimiter~forbidden.\
9322     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9323     because~the~key~'small'~is~in~force.\
9324     This~error~is~fatal.
9325 }
9326 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9327 {
9328     Unknown~cell.\
9329     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9330     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9331     can't~be~executed~because~a~cell~doesn't~exist.\
9332     This~command~\token_to_str:N \line\ will~be~ignored.
9333 }
9334 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9335 {
9336     Duplicate~name.\
9337     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9338     in~this~\@@_full_name_env:.\
9339     This~key~will~be~ignored.\
9340     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9341     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9342 }
9343 {
9344     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9345     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9346 }
9347 \@@_msg_new:nn { r~or~l~with~preamble }
9348 {
9349     Erroneous~use.\
9350     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9351     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9352     your~\@@_full_name_env:.\
9353     This~key~will~be~ignored.
9354 }
9355 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9356 {
9357     Erroneous~use.\
9358     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9359     the~array.~This~error~is~fatal.
9360 }
9361 \@@_msg_new:nn { bad~corner }
9362 {
9363     Bad~corner.\
9364     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9365     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
9366     This~specification~of~corner~will~be~ignored.
9367 }
9368 \@@_msg_new:nn { bad~border }
9369 {
9370     Bad~border.\
9371     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9372     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9373     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~

```

```

9374     also~use~the~key~'tikz'
9375     \IfPackageLoadedTF { tikz }
9376     { }
9377     {~if~you~load~the~LaTeX~package~'tikz'}).\
9378     This~specification~of~border~will~be~ignored.
9379 }

9380 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9381 {
9382     TikZ~not~loaded.\
9383     You~can't~use~\token_to_str:N \TikzEveryCell\
9384     because~you~have~not~loaded~tikz.~
9385     This~command~will~be~ignored.
9386 }

9387 \@@_msg_new:nn { tikz~key~without~tikz }
9388 {
9389     TikZ~not~loaded.\
9390     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9391     \Block'~because~you~have~not~loaded~tikz.~
9392     This~key~will~be~ignored.
9393 }

9394 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
9395 {
9396     Erroneous~use.\
9397     In~the~\@@_full_name_env:,~you~must~use~the~key~
9398     'last-col'~without~value.\
9399     However,~you~can~go~on~for~this~time~
9400     (the~value~'\l_keys_value_tl'~will~be~ignored).
9401 }

9402 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
9403 {
9404     Erroneous~use.\
9405     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9406     'last-col'~without~value.\
9407     However,~you~can~go~on~for~this~time~
9408     (the~value~'\l_keys_value_tl'~will~be~ignored).
9409 }

9410 \@@_msg_new:nn { Block~too~large~1 }
9411 {
9412     Block~too~large.\
9413     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9414     too~small~for~that~block. \
9415 }

9416 \@@_msg_new:nn { Block~too~large~2 }
9417 {
9418     Block~too~large.\
9419     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9420     \g_@@_static_num_of_col_int\
9421     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9422     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9423     (&)~at~the~end~of~the~first~row~of~your~
9424     \@@_full_name_env:.\
9425     This~block~and~maybe~others~will~be~ignored.
9426 }

9427 \@@_msg_new:nn { unknown~column~type }
9428 {
9429     Bad~column~type.\
9430     The~column~type~'#1'~in~your~\@@_full_name_env:\
9431     is~unknown. \
9432     This~error~is~fatal.
9433 }

```

```

9434 \@@_msg_new:nn { unknown~column~type~S }
9435 {
9436   Bad~column~type.\\
9437   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9438   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9439   load~that~package. \\
9440   This~error~is~fatal.
9441 }
9442 \@@_msg_new:nn { tabularnote~forbidden }
9443 {
9444   Forbidden~command.\\
9445   You~can't~use~the~command~\token_to_str:N\tabularnote\
9446   ~here.~This~command~is~available~only~in~
9447   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9448   the~argument~of~a~command~\token_to_str:N \caption\ included~
9449   in~an~environment~{table}. \\
9450   This~command~will~be~ignored.
9451 }
9452 \@@_msg_new:nn { borders~forbidden }
9453 {
9454   Forbidden~key.\\
9455   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9456   because~the~option~'rounded~corners'~
9457   is~in~force~with~a~non~zero~value.\\
9458   This~key~will~be~ignored.
9459 }
9460 \@@_msg_new:nn { bottomrule~without~booktabs }
9461 {
9462   booktabs~not~loaded.\\
9463   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9464   loaded~'booktabs'.\\
9465   This~key~will~be~ignored.
9466 }
9467 \@@_msg_new:nn { enumitem~not~loaded }
9468 {
9469   enumitem~not~loaded.\\
9470   You~can't~use~the~command~\token_to_str:N\tabularnote\
9471   ~because~you~haven't~loaded~'enumitem'.\\
9472   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9473   ignored~in~the~document.
9474 }
9475 \@@_msg_new:nn { tikz~without~tikz }
9476 {
9477   Tikz~not~loaded.\\
9478   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9479   loaded.~If~you~go~on,~that~key~will~be~ignored.
9480 }
9481 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9482 {
9483   Tikz~not~loaded.\\
9484   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9485   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9486   You~can~go~on~but~you~will~have~another~error~if~you~actually~
9487   use~that~custom~line.
9488 }
9489 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9490 {
9491   Tikz~not~loaded.\\
9492   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9493   command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9494   That~key~will~be~ignored.

```

```

9495 }
9496 \@@_msg_new:nn { without~color~inside }
9497 {
9498   If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9499   \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9500   outside~\token_to_str:N \CodeBefore,~you~
9501   should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\
9502   You~can~go~on~but~you~may~need~more~compilations.
9503 }
9504 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9505 {
9506   Erroneous~use.\
9507   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9508   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9509   The~key~'color'~will~be~discarded.
9510 }
9511 \@@_msg_new:nn { Wrong~last~row }
9512 {
9513   Wrong~number.\
9514   You~have~used~'last~row'=\int_use:N \l_@@_last_row_int'~but~your~
9515   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9516   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9517   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9518   without~value~(more~compilations~might~be~necessary).
9519 }
9520 \@@_msg_new:nn { Yet~in~env }
9521 {
9522   Nested~environments.\
9523   Environments~of~nicematrix~can't~be~nested.\
9524   This~error~is~fatal.
9525 }
9526 \@@_msg_new:nn { Outside~math~mode }
9527 {
9528   Outside~math~mode.\
9529   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9530   (and~not~in~\token_to_str:N \vcenter).\
9531   This~error~is~fatal.
9532 }
9533 \@@_msg_new:nn { One~letter~allowed }
9534 {
9535   Bad~name.\
9536   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
9537   It~will~be~ignored.
9538 }
9539 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9540 {
9541   Environment~{TabularNote}~forbidden.\
9542   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9543   but~*before*~the~\token_to_str:N \CodeAfter.\
9544   This~environment~{TabularNote}~will~be~ignored.
9545 }
9546 \@@_msg_new:nn { varwidth~not~loaded }
9547 {
9548   varwidth~not~loaded.\
9549   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9550   loaded.\
9551   Your~column~will~behave~like~'p'.
9552 }
9553 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9554 {

```

```

9555     Unknown~key.\\
9556     Your~key~'\l_keys_key_str'~is-unknown~for~a~rule.\\
9557     \c_@@_available_keys_str
9558 }
9559 {
9560     The~available~keys~are~(in~alphabetic~order):~
9561     color,~
9562     dotted,~
9563     multiplicity,~
9564     sep-color,~
9565     tikz,~and~total~width.
9566 }
9567
9568 \@@_msg_new:nnn { Unknown~key~for~Block }
9569 {
9570     Unknown~key.\\
9571     The~key~'\l_keys_key_str'~is-unknown~for~the~command~\token_to_str:N
9572     \Block.\\ It~will~be~ignored. \\
9573     \c_@@_available_keys_str
9574 }
9575 {
9576     The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9577     hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9578     respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9579 }
9580 \@@_msg_new:nnn { Unknown~key~for~Brace }
9581 {
9582     Unknown~key.\\
9583     The~key~'\l_keys_key_str'~is-unknown~for~the~commands~\token_to_str:N
9584     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9585     It~will~be~ignored. \\
9586     \c_@@_available_keys_str
9587 }
9588 {
9589     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9590     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9591     right~shorten)~and~yshift.
9592 }
9593 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9594 {
9595     Unknown~key.\\
9596     The~key~'\l_keys_key_str'~is-unknown.\\
9597     It~will~be~ignored. \\
9598     \c_@@_available_keys_str
9599 }
9600 {
9601     The~available~keys~are~(in~alphabetic~order):~
9602     delimiters/color,~
9603     rules~(with~the~subkeys~'color'~and~'width'),~
9604     sub~matrix~(several~subkeys)~
9605     and~xdots~(several~subkeys).~
9606     The~latter~is~for~the~command~\token_to_str:N \line.
9607 }
9608 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9609 {
9610     Unknown~key.\\
9611     The~key~'\l_keys_key_str'~is-unknown.\\
9612     It~will~be~ignored. \\
9613     \c_@@_available_keys_str
9614 }
9615 {
9616     The~available~keys~are~(in~alphabetic~order):~

```

```

9617     create-cell-nodes,~
9618     delimiters/color~and~
9619     sub-matrix~(several~subkeys).
9620 }
9621 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9622 {
9623     Unknown~key.\\
9624     The~key~'\l_keys_key_str'~is~unknown.\\
9625     That~key~will~be~ignored. \\
9626     \c_@@_available_keys_str
9627 }
9628 {
9629     The~available~keys~are~(in~alphabetic~order):~
9630     'delimiters/color',~
9631     'extra-height',~
9632     'hlines',~
9633     'hvlines',~
9634     'left-xshift',~
9635     'name',~
9636     'right-xshift',~
9637     'rules'~(with~the~subkeys~'color'~and~'width'),~
9638     'slim',~
9639     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9640     and~'right-xshift').\\
9641 }
9642 \@@_msg_new:nnn { Unknown~key~for~notes }
9643 {
9644     Unknown~key.\\
9645     The~key~'\l_keys_key_str'~is~unknown.\\
9646     That~key~will~be~ignored. \\
9647     \c_@@_available_keys_str
9648 }
9649 {
9650     The~available~keys~are~(in~alphabetic~order):~
9651     bottomrule,~
9652     code-after,~
9653     code-before,~
9654     detect-duplicates,~
9655     enumitem-keys,~
9656     enumitem-keys-para,~
9657     para,~
9658     label-in-list,~
9659     label-in-tabular~and~
9660     style.
9661 }
9662 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9663 {
9664     Unknown~key.\\
9665     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9666     \token_to_str:N \RowStyle. \\
9667     That~key~will~be~ignored. \\
9668     \c_@@_available_keys_str
9669 }
9670 {
9671     The~available~keys~are~(in~alphabetic~order):~
9672     'bold',~
9673     'cell-space-top-limit',~
9674     'cell-space-bottom-limit',~
9675     'cell-space-limits',~
9676     'color',~
9677     'nb-rows'~and~
9678     'rowcolor'.
9679 }

```

```

9680 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9681 {
9682   Unknown~key.\\
9683   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9684   \token_to_str:N \NiceMatrixOptions. \\
9685   That~key~will~be~ignored. \\
9686   \c_@@_available_keys_str
9687 }
9688 {
9689   The~available~keys~are~(in~alphabetic~order):~
9690   allow~duplicate~names,~
9691   caption~above,~
9692   cell~space~bottom~limit,~
9693   cell~space~limits,~
9694   cell~space~top~limit,~
9695   code~for~first~col,~
9696   code~for~first~row,~
9697   code~for~last~col,~
9698   code~for~last~row,~
9699   corners,~
9700   custom~key,~
9701   create~extra~nodes,~
9702   create~medium~nodes,~
9703   create~large~nodes,~
9704   delimiters~(several~subkeys),~
9705   end~of~row,~
9706   first~col,~
9707   first~row,~
9708   hlines,~
9709   hvlines,~
9710   hvlines~except~borders,~
9711   last~col,~
9712   last~row,~
9713   left~margin,~
9714   light~syntax,~
9715   matrix/columns~type,~
9716   notes~(several~subkeys),~
9717   nullify~dots,~
9718   pgf~node~code,~
9719   renew~dots,~
9720   renew~matrix,~
9721   respect~arraystretch,~
9722   rounded~corners,~
9723   right~margin,~
9724   rules~(with~the~subkeys~'color'~and~'width'),~
9725   small,~
9726   sub~matrix~(several~subkeys),~
9727   vlimes,~
9728   xdots~(several~subkeys).
9729 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no 1 and r.

```

9730 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9731 {
9732   Unknown~key.\\
9733   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9734   \{NiceArray\}. \\
9735   That~key~will~be~ignored. \\
9736   \c_@@_available_keys_str
9737 }
9738 {
9739   The~available~keys~are~(in~alphabetic~order):~
9740   b,~

```

```

9741 baseline,~
9742 c,~
9743 cell-space-bottom-limit,~
9744 cell-space-limits,~
9745 cell-space-top-limit,~
9746 code-after,~
9747 code-for-first-col,~
9748 code-for-first-row,~
9749 code-for-last-col,~
9750 code-for-last-row,~
9751 color-inside,~
9752 columns-width,~
9753 corners,~
9754 create-extra-nodes,~
9755 create-medium-nodes,~
9756 create-large-nodes,~
9757 extra-left-margin,~
9758 extra-right-margin,~
9759 first-col,~
9760 first-row,~
9761 hlines,~
9762 hvlines,~
9763 hvlines-except-borders,~
9764 last-col,~
9765 last-row,~
9766 left-margin,~
9767 light-syntax,~
9768 name,~
9769 nullify-dots,~
9770 pgf-node-code,~
9771 renew-dots,~
9772 respect-arraystretch,~
9773 right-margin,~
9774 rounded-corners,~
9775 rules~(with~the~subkeys~'color'~and~'width'),~
9776 small,~
9777 t,~
9778 vlines,~
9779 xdots/color,~
9780 xdots/shorten-start,~
9781 xdots/shorten-end,~
9782 xdots/shorten~and~
9783 xdots/line-style.
9784 }

```

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`).

```

9785 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9786 {
9787   Unknown~key.\\
9788   The~key~'\l_keys_key_str'~is~unknown~for~the~
9789   \@@_full_name_env:. \\
9790   That~key~will~be~ignored. \\
9791   \c_@@_available_keys_str
9792 }
9793 {
9794   The~available~keys~are~(in~alphabetic~order):~
9795   b,~
9796   baseline,~
9797   c,~
9798   cell-space-bottom-limit,~
9799   cell-space-limits,~
9800   cell-space-top-limit,~
9801   code-after,~

```



```

9802 code-for-first-col,~
9803 code-for-first-row,~
9804 code-for-last-col,~
9805 code-for-last-row,~
9806 color-inside,~
9807 columns-type,~
9808 columns-width,~
9809 corners,~
9810 create-extra-nodes,~
9811 create-medium-nodes,~
9812 create-large-nodes,~
9813 extra-left-margin,~
9814 extra-right-margin,~
9815 first-col,~
9816 first-row,~
9817 hlines,~
9818 hvlines,~
9819 hvlines-except-borders,~
9820 l,~
9821 last-col,~
9822 last-row,~
9823 left-margin,~
9824 light-syntax,~
9825 name,~
9826 nullify-dots,~
9827 pgf-node-code,~
9828 r,~
9829 renew-dots,~
9830 respect-arraystretch,~
9831 right-margin,~
9832 rounded-corners,~
9833 rules~(with~the~subkeys~'color'~and~'width'),~
9834 small,~
9835 t,~
9836 vlines,~
9837 xdots/color,~
9838 xdots/shorten-start,~
9839 xdots/shorten-end,~
9840 xdots/shorten-and~
9841 xdots/line-style.
9842 }
9843 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9844 {
9845   Unknown~key.\\
9846   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9847   \{NiceTabular\}. \\
9848   That~key~will~be~ignored. \\
9849   \c_@@_available_keys_str
9850 }
9851 {
9852   The~available~keys~are~(in~alphabetic~order):~
9853   b,~
9854   baseline,~
9855   c,~
9856   caption,~
9857   cell-space-bottom-limit,~
9858   cell-space-limits,~
9859   cell-space-top-limit,~
9860   code-after,~
9861   code-for-first-col,~
9862   code-for-first-row,~
9863   code-for-last-col,~
9864   code-for-last-row,~

```

```

9865 color-inside,~
9866 columns-width,~
9867 corners,~
9868 custom-line,~
9869 create-extra-nodes,~
9870 create-medium-nodes,~
9871 create-large-nodes,~
9872 extra-left-margin,~
9873 extra-right-margin,~
9874 first-col,~
9875 first-row,~
9876 hlines,~
9877 hvlines,~
9878 hvlines-except-borders,~
9879 label,~
9880 last-col,~
9881 last-row,~
9882 left-margin,~
9883 light-syntax,~
9884 name,~
9885 notes~(several-subkeys),~
9886 nullify-dots,~
9887 pgf-node-code,~
9888 renew-dots,~
9889 respect-arraystretch,~
9890 right-margin,~
9891 rounded-corners,~
9892 rules~(with~the~subkeys~'color'~and~'width'),~
9893 short-caption,~
9894 t,~
9895 tabularnote,~
9896 vlines,~
9897 xdots/color,~
9898 xdots/shorten-start,~
9899 xdots/shorten-end,~
9900 xdots/shorten-and~
9901 xdots/line-style.
9902 }

9903 \@@_msg_new:nnn { Duplicate-name }
9904 {
9905   Duplicate-name.\\
9906   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9907   the~same~environment~name~twice.~You~can~go~on,~but,~
9908   maybe,~you~will~have~incorrect~results~especially~
9909   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9910   message~again,~use~the~key~'allow-duplicate-names'~in~
9911   '\token_to_str:N \NiceMatrixOptions'.\\
9912   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9913     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9914 }
9915 {
9916   The~names~already~defined~in~this~document~are:~
9917   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9918 }

9919 \@@_msg_new:nn { Option-auto-for-columns-width }
9920 {
9921   Erroneous-use.\\
9922   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9923   That~key~will~be~ignored.
9924 }

9925 \@@_msg_new:nn { NiceTabularX-without-X }
9926 {
9927   NiceTabularX-without-X.\\

```

```

9928     You~should~not~use~{NiceTabularX}~without~X~columns.\\
9929     However,~you~can~go~on.
9930 }

9931 \@@_msg_new:nn { Preamble~forgotten }
9932 {
9933     Preamble~forgotten.\\
9934     You~have~probably~forgotten~the~preamble~of~your~
9935     \@@_full_name_env:. \\
9936     This~error~is~fatal.
9937 }

```

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	57
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	127
23	The vertical and horizontal rules	137
24	The empty corners	152
25	The environment <code>{NiceMatrixBlock}</code>	155
26	The extra nodes	156
27	The blocks	160
28	How to draw the dotted lines transparently	180
29	Automatic arrays	180
30	The redefinition of the command <code>\dotfill</code>	181

31	The command <code>\diagbox</code>	182
32	The keyword <code>\CodeAfter</code>	183
33	The delimiters in the preamble	184
34	The command <code>\SubMatrix</code>	185
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	193
36	The command <code>TikzEveryCell</code>	196
37	The command <code>\ShowCellNames</code>	198
38	We process the options at package loading	200
39	About the package underscore	202
40	Error messages of the package	202