

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

November 16, 2023

Abstract

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

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

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

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

We load some packages.

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

*This document corresponds to the version 6.25b of `nicematrix`, at the date of 2023/11/16.

```

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 then 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 contain 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 distincts environments of `nicematrix` (these names are global and not local to the current TeX scope). However, the option `allow-duplicate-names` disables this feature.

```

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_t1` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of `array`) with the option `t` and the right translation will be done further. Remark that `\str_if_eq:VnTF` is fully expandable and we need something fully expandable here.

```

1429 [ \str_if_eq:VnTF \l_@@_baseline_t1 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 }
1611 \@@_revert_colortbl:

```

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

```

1612 \tl_if_exist:NT \l_@@_note_in_caption_tl
1613 {
1614   \tl_if_empty:NF \l_@@_note_in_caption_tl
1615   {
1616     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1617     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1618   }
1619 }

```

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

```

1620 \seq_gclear:N \g_@@_multicolumn_cells_seq
1621 \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).

```

1622 \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`).

```
1623 \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.

```
1624 \int_gzero_new:N \g_@@_col_total_int
1625 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1626 \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.

```
1627 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1628 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1629 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1630 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1631 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1632 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1633 \tl_gclear:N \g_nicematrix_code_before_tl
1634 \tl_gclear:N \g_@@_pre_code_before_tl
1635 }
```

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

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

```
1636 \cs_new_protected:Npn \@@_pre_array:
1637 {
1638   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1639   \int_gzero_new:N \c@iRow
1640   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1641   \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).

```
1642 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1643 {
1644   \bool_set_true:N \l_@@_last_row_without_value_bool
1645   \bool_if:NT \g_@@_aux_found_bool
1646   { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1647 }
1648 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1649 {
1650   \bool_if:NT \g_@@_aux_found_bool
1651   { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1652 }
```

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”.

```
1653 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1654 {
1655   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1656   {
1657     \dim_gset:Nn \g_@@_ht_last_row_dim
1658     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1659     \dim_gset:Nn \g_@@_dp_last_row_dim
1660     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1661   }
1662 }
```

```

1663 \seq_gclear:N \g_@@_cols_vlism_seq
1664 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1665 \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.

```

1666 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1667 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1668 \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.

```

1669 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `−2` is important.

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

```

1670 \@@_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.

```

1671 \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]`).

```

1672 \dim_zero_new:N \l_@@_left_delim_dim
1673 \dim_zero_new:N \l_@@_right_delim_dim
1674 \bool_if:NTF \g_@@_delims_bool
1675 {

```

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

```

1676 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1677 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1678 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1679 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1680 }
1681 {
1682 \dim_gset:Nn \l_@@_left_delim_dim
1683 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1684 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1685 }

```

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).

```

1686 \hbox_set:Nw \l_@@_the_array_box
1687 \skip_horizontal:N \l_@@_left_margin_dim
1688 \skip_horizontal:N \l_@@_extra_left_margin_dim
1689 \c_math_toggle_token
1690 \bool_if:NTF \l_@@_light_syntax_bool
1691 { \use:c { @@-light-syntax } }
1692 { \use:c { @@-normal-syntax } }
1693 }

```


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

```

1694 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1695 {
1696   \tl_set:Nn \l_tmpa_tl { #1 }
1697   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1698     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1699   \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1700   \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...

```

1701   \@@_pre_array:
1702 }

```

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).

```

1703 \cs_new_protected:Npn \@@_pre_code_before:
1704 {

```

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).

```

1705   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1706   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1707   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1708   \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.

```

1709   \pgfsys@markposition { \@@_env: - position }
1710   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1711   \pgfpicture
1712   \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1713   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1714   {
1715     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1716     \pgfcoordinate { \@@_env: - row - ##1 }
1717     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1718   }

```

Now, the recreation of the `col` nodes.

```

1719   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1720   {
1721     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1722     \pgfcoordinate { \@@_env: - col - ##1 }
1723     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1724   }

```

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

```

1725   \@@_create_diag_nodes:

```

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

```
1726 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1727 \endpgfpicture
```

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

```
1728 \@@_create_blocks_nodes:
1729 \IfPackageLoadedTF { tikz }
1730 {
1731   \tikzset
1732   {
1733     every~picture / .style =
1734     { overlay , name~prefix = \@@_env: - }
1735   }
1736 }
1737 { }
1738 \cs_set_eq:NN \cellcolor \@@_cellcolor
1739 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1740 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1741 \cs_set_eq:NN \rowcolor \@@_rowcolor
1742 \cs_set_eq:NN \rowcolors \@@_rowcolors
1743 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1744 \cs_set_eq:NN \arraycolor \@@_arraycolor
1745 \cs_set_eq:NN \columncolor \@@_columncolor
1746 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1747 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1748 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1749 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1750 }
```

```
1751 \cs_new_protected:Npn \@@_exec_code_before:
1752 {
1753   \seq_gclear_new:N \g_@@_colors_seq
1754   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1755   \group_begin:
```

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

```
1756 \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`).

```
1757 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1758 { \@@_rescan_for_spanish:N \l_@@_code_before_tl }
```

Here is the `\CodeBefore`. The construction is a bit complicated because `\g_@@_pre_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\g_@@_pre_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That’s why we use a `\q_stop`: it will be used to discard the rest of `\g_@@_pre_code_before_tl`.

```
1759 \exp_last_unbraced:Nv \@@_CodeBefore_keys:
1760 \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.

```
1761 \@@_actually_color:
1762 \l_@@_code_before_tl
1763 \q_stop
1764 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
```

```

1765 \group_end:
1766 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1767 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1768 }

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

1779 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1780 {
1781   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1782   \@@_CodeBefore:w
1783 }

```

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.

```

1784 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1785 {
1786   \bool_if:NT \g_@@_aux_found_bool
1787   {
1788     \@@_pre_code_before:
1789     #1
1790   }
1791 }

```

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.

```

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

```

```

1817 \int_step_inline:nn \c@iRow
1818 {
1819     \pgfnodealias
1820     { \@@_env: - ##1 - last }
1821     { \@@_env: - ##1 - \int_use:N \c@jCol }
1822 }
1823 \int_step_inline:nn \c@jCol
1824 {
1825     \pgfnodealias
1826     { \@@_env: - last - ##1 }
1827     { \@@_env: - \int_use:N \c@iRow - ##1 }
1828 }
1829 \@@_create_extra_nodes:
1830 }

```

```

1831 \cs_new_protected:Npn \@@_create_blocks_nodes:
1832 {
1833     \pgfpicture
1834     \pgf@relevantforpicturesizefalse
1835     \pgfrememberpicturepositiononpagetrue
1836     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1837     { \@@_create_one_block_node:nnnnn ##1 }
1838     \endpgfpicture
1839 }

```

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

```

1840 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1841 {
1842     \tl_if_empty:nF { #5 }
1843     {
1844         \@@_qpoint:n { col - #2 }
1845         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1846         \@@_qpoint:n { #1 }
1847         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1848         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1849         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1850         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1851         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1852         \@@_pgf_rect_node:nnnnn
1853         { \@@_env: - #5 }
1854         { \dim_use:N \l_tmpa_dim }
1855         { \dim_use:N \l_tmpb_dim }
1856         { \dim_use:N \l_@@_tmpc_dim }
1857         { \dim_use:N \l_@@_tmpd_dim }
1858     }
1859 }

```

```

1860 \cs_new_protected:Npn \@@_patch_for_revtext:
1861 {
1862     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1863     \cs_set_eq:NN \insert@column \insert@column@array
1864     \cs_set_eq:NN \@classx \@classx@array
1865     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1866     \cs_set_eq:NN \@arraycr \@arraycr@array
1867     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1868     \cs_set_eq:NN \array \array@array
1869     \cs_set_eq:NN \@array \@array@array

```

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).

```

1870 \cs_set_eq:NN \@tabular \@tabular@array
1871 \cs_set_eq:NN \@mkpream \@mkpream@array
1872 \cs_set_eq:NN \endarray \endarray@array
1873 \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1874 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1875 }

```

11 The environment {NiceArrayWithDelims}

```

1876 \NewDocumentEnvironment { NiceArrayWithDelims }
1877 { m m O { } m ! O { } t \CodeBefore }
1878 {
1879 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1880 \@@_provide_pgfsyspdfmark:
1881 \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.

```

1882 \bgroup

1883 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1884 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1885 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1886 \int_gzero:N \g_@@_block_box_int
1887 \dim_zero:N \g_@@_width_last_col_dim
1888 \dim_zero:N \g_@@_width_first_col_dim
1889 \bool_gset_false:N \g_@@_row_of_col_done_bool
1890 \str_if_empty:NT \g_@@_name_env_str
1891 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1892 \bool_if:NTF \l_@@_tabular_bool
1893 \mode_leave_vertical:
1894 \@@_test_if_math_mode:
1895 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1896 \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.

```

1897 \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.

```

1898 \cs_if_exist:NT \tikz@library@external@loaded
1899 {
1900 \tikzexternaldisable
1901 \cs_if_exist:NT \ifstandalone
1902 { \tikzset { external / optimize = false } }
1903 }

```

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

```

1904 \int_gincr:N \g_@@_env_int
1905 \bool_if:NF \l_@@_block_auto_columns_width_bool

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```
1906 { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

The sequence `\g_@@_blocks_seq` will contain the characteristics 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`).

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

```
1908 \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`.

```
1909 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
```

```
1910 \seq_gclear:N \g_@@_pos_of_xdots_seq
```

```
1911 \tl_gclear_new:N \g_@@_code_before_tl
```

```
1912 \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.

```
1913 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
```

```
1914 {
```

```
1915   \bool_gset_true:N \g_@@_aux_found_bool
```

```
1916   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
```

```
1917 }
```

```
1918 { \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.

```
1919 \tl_build_gbegin:N \g_@@_aux_tl
```

```
1920 \tl_if_empty:NF \g_@@_code_before_tl
```

```
1921 {
```

```
1922   \bool_set_true:N \l_@@_code_before_bool
```

```
1923   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
```

```
1924 }
```

```
1925 \tl_if_empty:NF \g_@@_pre_code_before_tl
```

```
1926 { \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`.

```
1927 \bool_if:NTF \g_@@_delims_bool
```

```
1928 { \keys_set:nn { NiceMatrix / pNiceArray } }
```

```
1929 { \keys_set:nn { NiceMatrix / NiceArray } }
```

```
1930 { #3 , #5 }
```

```
1931 \@@_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:`.

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

```
1933 }
```

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

```
1934 {
```

```
1935   \bool_if:NTF \l_@@_light_syntax_bool
```

```
1936   { \use:c { end @@-light-syntax } }
```

```
1937   { \use:c { end @@-normal-syntax } }
```

```
1938   \c_math_toggle_token
```

```
1939   \skip_horizontal:N \l_@@_right_margin_dim
```

```
1940   \skip_horizontal:N \l_@@_extra_right_margin_dim
```

```
1941   \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.

```

1942 \bool_if:NT \l_@@_width_used_bool
1943 {
1944   \int_if_zero:nT \g_@@_total_X_weight_int
1945   { \@@_error_or_warning:n { width-without~X-columns } }
1946 }

```

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 .

```

1947 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1948 {
1949   \tl_build_gput_right:Nx \g_@@_aux_tl
1950   {
1951     \bool_set_true:N \l_@@_X_columns_aux_bool
1952     \dim_set:Nn \l_@@_X_columns_dim
1953     {
1954       \dim_compare:nNnTF
1955       {
1956         \dim_abs:n
1957         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1958       }
1959       <
1960       { 0.001 pt }
1961       { \dim_use:N \l_@@_X_columns_dim }
1962       {
1963         \dim_eval:n
1964         {
1965           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1966           / \int_use:N \g_@@_total_X_weight_int
1967           + \l_@@_X_columns_dim
1968         }
1969       }
1970     }
1971   }
1972 }

```

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

```

1973 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1974 {
1975   \bool_if:NF \l_@@_last_row_without_value_bool
1976   {
1977     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1978     {
1979       \@@_error:n { Wrong~last~row }
1980       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1981     }
1982   }
1983 }

```

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

```

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

```

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

```
1990     }
```

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

```
1991     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1992     \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).

```
1993     \int_if_zero:nT \l_@@_first_col_int
1994     {
1995         % \skip_horizontal:N \col@sep % 05-08-23
1996         \skip_horizontal:N \g_@@_width_first_col_dim
1997     }
```

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

```
1998     \bool_if:nTF { ! \g_@@_delims_bool }
1999     {
2000         \str_case:VnF \l_@@_baseline_tl
2001         {
2002             b \@@_use_arraybox_with_notes_b:
2003             c \@@_use_arraybox_with_notes_c:
2004         }
2005         \@@_use_arraybox_with_notes:
2006     }
```

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).

```
2007     {
2008         \int_if_zero:nTF \l_@@_first_row_int
2009         {
2010             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2011             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2012         }
2013         { \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

```
2014     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2015     {
2016         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2017         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2018     }
2019     { \dim_zero:N \l_tmpb_dim }
2020     \hbox_set:Nn \l_tmpa_box
2021     {
2022         \c_math_toggle_token
2023         \@@_color:V \l_@@_delimiters_color_tl
2024         \exp_after:wN \left \g_@@_left_delim_tl
2025         \vcenter
2026         {
```

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.

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

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).


```

2035         { \skip_horizontal:N -\tabcolsep }
2036         { \skip_horizontal:N -\arraycolsep }
2037     }

```

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

```

2038         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
2039     }

```

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.

```

2040         \@@_color:V \l_@@_delimiters_color_tl
2041         \exp_after:wN \right \g_@@_right_delim_tl
2042         \c_math_toggle_token
2043     }

```

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.

```

2044         \bool_if:NTF \l_@@_delimiters_max_width_bool
2045         {
2046             \@@_put_box_in_flow_bis:nn
2047             \g_@@_left_delim_tl \g_@@_right_delim_tl
2048         }
2049         \@@_put_box_in_flow:
2050     }

```

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).

```

2051         \bool_if:NT \g_@@_last_col_found_bool
2052         {
2053             \skip_horizontal:N \g_@@_width_last_col_dim
2054             % \skip_horizontal:N \col@sep % 2023-08-05
2055         }
2056         \bool_if:NT \l_@@_preamble_bool
2057         {
2058             \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2059             { \@@_warning_gredirect_none:n { columns-not-used } }
2060         }
2061         \@@_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.

```

2062         \egroup

```

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

```

2063         \tl_build_gend:N \g_@@_aux_tl
2064         \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2065         \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2066         \iow_now:Nx \@mainaux
2067         {
2068             \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2069             { \exp_not:V \g_@@_aux_tl }
2070         }
2071         \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2072         \bool_if:NT \g_@@_footnote_bool \endsavenotes
2073     }

```

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.

```

2074 \cs_new_protected:Npn \@@_transform_preamble:
2075 {
2076   \@@_transform_preamble_i:
2077   \@@_transform_preamble_ii:
2078 }
2079 \cs_new_protected:Npn \@@_transform_preamble_i:
2080 {
2081   \int_gzero:N \c@jCol

```

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

```

2082 \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.

```

2083 \bool_gset_false:N \g_tmpb_bool

```

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

```

2084 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2085 \int_zero:N \l_tmpa_int
2086 \tl_gclear:N \g_@@_array_preamble_tl
2087 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2088 {
2089   \tl_gset:Nn \g_@@_array_preamble_tl
2090   { ! { \skip_horizontal:N \arrayrulewidth } }
2091 }
2092 {
2093   \clist_if_in:NnT \l_@@_vlines_clist 1
2094   {
2095     \tl_gset:Nn \g_@@_array_preamble_tl
2096     { ! { \skip_horizontal:N \arrayrulewidth } }
2097   }
2098 }

```

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

```

2099 \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2100 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

```

2101 \@@_replace_columncolor:
2102 }

```

```

2103 \hook_gput_code:nnn { begindocument } { . }
2104 {
2105   \IfPackageLoadedTF { colortbl }
2106   {
2107     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2108     \cs_new_protected:Npn \@@_replace_columncolor:
2109     {
2110       \regex_replace_all:NnN
2111       \c_@@_columncolor_regex
2112       { \c { @@_columncolor_preamble } }

```

```

2113         \g_@@_array_preamble_tl
2114     }
2115 }
2116 {
2117     \cs_new_protected:Npn \@@_replace_columncolor:
2118     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2119 }
2120 }

```

```

2121 \cs_new_protected:Npn \@@_transform_preamble_ii:
2122 {

```

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

```

2123     \bool_lazy_or:nnT
2124     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
2125     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2126     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2127     \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).

```

2128     \int_if_zero:nTF \l_@@_first_col_int
2129     { \tl_gput_left:NV \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2130     {
2131         \bool_lazy_all:nT
2132         {
2133             { \bool_not_p:n \g_@@_delims_bool }
2134             { \bool_not_p:n \l_@@_tabular_bool }
2135             { \tl_if_empty_p:N \l_@@_vlines_clist }
2136             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2137         }
2138         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2139     }
2140     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2141     { \tl_gput_right:NV \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2142     {
2143         \bool_lazy_all:nT
2144         {
2145             { \bool_not_p:n \g_@@_delims_bool }
2146             { \bool_not_p:n \l_@@_tabular_bool }
2147             { \tl_if_empty_p:N \l_@@_vlines_clist }
2148             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2149         }
2150         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2151     }

```

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`).

```

2152     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2153     {
2154         \tl_gput_right:Nn \g_@@_array_preamble_tl
2155         { > { \@@_error_too_much_cols: } 1 }
2156     }
2157 }

```

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

```

2158 \cs_new_protected:Npn \@@_rec_preamble:n #1
2159 {

```

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

```
2160 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2161 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2162 { }
```

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

```
2163 \cs_if_exist:cTF { NC @ find @ #1 }
2164 {
2165   \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2166   \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2167 }
2168 {
2169   \tl_if_eq:nnT { #1 } { S }
2170   { \@@_fatal:n { unknown~column~type~S } }
2171   { \@@_fatal:nn { unknown~column~type } { #1 } }
2172 }
2173 }
2174 }
```

For c, l and r

```
2175 \cs_new:Npn \@@_c #1
2176 {
2177   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2178   \tl_gclear:N \g_@@_pre_cell_tl
2179   \tl_gput_right:Nn \g_@@_array_preamble_tl
2180   {
2181     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2182     #1
2183     < \@@_cell_end:
2184   }
}
```

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

```
2185 \int_gincr:N \c@jCol
2186 \@@_rec_preamble_after_col:n
2187 }
2188 \cs_set_eq:NN \@@_l \@@_c
2189 \cs_set_eq:NN \@@_r \@@_c
```

For ! and @

```
2190 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2191 {
2192   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2193   \@@_rec_preamble:n
2194 }
2195 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }
```

For |

```
2196 \cs_new:cpn { @@ _ | } #1
2197 { }
```

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

```
2198 \int_incr:N \l_tmpa_int
2199 \@@_make_preamble_i_i:n
2200 }
```

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`.

```

2201 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2202 {
2203   \str_if_eq:nnTF { #1 } |
2204     { \use:c { @@ _ | } | }
2205     { \@@_make_preamble_i_ii:nn { } #1 }
2206 }
2207 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2208 {
2209   \str_if_eq:nnTF { #2 } [
2210     { \@@_make_preamble_i_iii:nw { #1 } [ }
2211     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2212   ]
2213 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2214 { \@@_make_preamble_i_iii:nn { #1 , #2 } }
2215 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2216 {
2217   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2218   \tl_gput_right:Nx \g_@@_array_preamble_tl
2219   {

```

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

```

2220     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2221   }
2222   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2223   {
2224     \@@_vline:n
2225     {
2226       position = \int_eval:n { \c_jCol + 1 } ,
2227       multiplicity = \int_use:N \l_tmpa_int ,
2228       total-width = \dim_use:N \l_@@_rule_width_dim ,
2229       #2
2230     }

```

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.

```

2231   }
2232   \int_zero:N \l_tmpa_int
2233   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2234   \@@_rec_preamble:n #1
2235 }
2236 \cs_new:cpn { @@ _ > } #1 #2
2237 {
2238   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2239   \@@_rec_preamble:n
2240 }
2241 \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.

```

2242 \keys_define:nn { WithArrows / p-column }
2243 {
2244   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2245   r .value_forbidden:n = true ,
2246   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2247   c .value_forbidden:n = true ,
2248   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2249   l .value_forbidden:n = true ,
2250   R .code:n =
2251     \IfPackageLoadedTF { ragged2e }
2252     { \str_set:Nn \l_@@_hpos_col_str { R } }

```

```

2253     {
2254       \@@_error_or_warning:n { ragged2e-not-loaded }
2255       \str_set:Nn \l_@@_hpos_col_str { r }
2256     } ,
2257     R .value_forbidden:n = true ,
2258     L .code:n =
2259       \IfPackageLoadedTF { ragged2e }
2260       { \str_set:Nn \l_@@_hpos_col_str { L } }
2261       {
2262         \@@_error_or_warning:n { ragged2e-not-loaded }
2263         \str_set:Nn \l_@@_hpos_col_str { l }
2264       } ,
2265     L .value_forbidden:n = true ,
2266     C .code:n =
2267       \IfPackageLoadedTF { ragged2e }
2268       { \str_set:Nn \l_@@_hpos_col_str { C } }
2269       {
2270         \@@_error_or_warning:n { ragged2e-not-loaded }
2271         \str_set:Nn \l_@@_hpos_col_str { c }
2272       } ,
2273     C .value_forbidden:n = true ,
2274     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2275     S .value_forbidden:n = true ,
2276     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2277     p .value_forbidden:n = true ,
2278     t .meta:n = p ,
2279     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2280     m .value_forbidden:n = true ,
2281     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2282     b .value_forbidden:n = true ,
2283   }

```

For p, b and m.

```

2284 \cs_new:Npn \@@_p #1
2285 {
2286   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2287   \@@_make_preamble_ii_i:n
2288 }
2289 \cs_set_eq:NN \@@_b \@@_p
2290 \cs_set_eq:NN \@@_m \@@_p
2291 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2292 {
2293   \str_if_eq:nnTF { #1 } { [ ]
2294     { \@@_make_preamble_ii_ii:w [ ] }
2295     { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2296   }
2297 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2298 { \@@_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.

```

2299 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2300 {

```

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).

```

2301   \str_set:Nn \l_@@_hpos_col_str { j }
2302   \tl_set:Nn \l_tmpa_tl { #1 }
2303   \@@_keys_p_column:V \l_tmpa_tl
2304   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2305 }

```

```

2306 \cs_new_protected:Npn \@@_keys_p_column:n #1
2307   { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2308 \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.

```

2309 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2310   {
2311     \use:e
2312     {
2313       \@@_make_preamble_ii_v:nnnnnnnn
2314       { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2315       { \dim_eval:n { #1 } }
2316     }

```

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.

```

2317       \str_if_eq:VnTF \l_@@_hpos_col_str j
2318       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2319       {
2320         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2321         { \str_lowercase:V \l_@@_hpos_col_str }
2322       }
2323       \str_case:Vn \l_@@_hpos_col_str
2324       {
2325         c { \exp_not:N \centering }
2326         l { \exp_not:N \raggedright }
2327         r { \exp_not:N \raggedleft }
2328         C { \exp_not:N \Centering }
2329         L { \exp_not:N \RaggedRight }
2330         R { \exp_not:N \RaggedLeft }
2331       }
2332       #3
2333     }
2334     { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2335     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2336     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2337     { #2 }
2338     {
2339       \str_case:VnF \l_@@_hpos_col_str
2340       {
2341         { j } { c }
2342         { si } { c }
2343       }

```

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

```

2344       { \str_lowercase:V \l_@@_hpos_col_str }
2345     }
2346   }

```

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

```

2347   \int_gincr:N \c@jCol
2348   \@@_rec_preamble_after_col:n
2349 }

```

#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*.

```

2350 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2351 {
2352   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2353     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2354     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2355   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2356   \tl_gclear:N \g_@@_pre_cell_tl
2357   \tl_gput_right:Nn \g_@@_array_preamble_tl
2358     {
2359       > {

```

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.

```

2360       \dim_set:Nn \l_@@_col_width_dim { #2 }
2361       \@@_cell_begin:w

```

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

```

2362       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2363       \everypar
2364       {
2365         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2366         \everypar { }
2367       }

```

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

```

2368       #3

```

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

```

2369       \g_@@_row_style_tl
2370       \arraybackslash
2371       #5
2372     }
2373     #8
2374     < {
2375       #6

```

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

```

2376       \@finalstrut \@arstrutbox
2377       \use:c { end #7 }

```

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

```

2378       #4
2379       \@@_cell_end:
2380     }
2381   }
2382 }

```

```

2383 \str_new:N \c_@@_ignorespaces_str
2384 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2385 \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...

```

2386 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2387 \cs_new_protected:Npn \@@_test_if_empty_i:
2388 {
2389   \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2390   \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2391   { \@@_test_if_empty:w }
2392 }
2393 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2394 {
2395   \peek_meaning:NT \unskip
2396   {
2397     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2398     {
2399       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2400       \skip_horizontal:N \l_@@_col_width_dim
2401     }
2402   }
2403 }
2404 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2405 {
2406   \peek_meaning:NT \_siunitx_table_skip:n
2407   {
2408     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2409     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2410   }
2411 }

```

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.

```

2412 \cs_new_protected:Npn \@@_center_cell_box:
2413 {

```

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

```

2414   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2415   {
2416     \int_compare:nNnT
2417     { \box_ht:N \l_@@_cell_box }
2418     >

```

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).

```

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

```

```

2431     }
2432 }

```

For V (similar to the V of varwidth).

```

2433 \cs_new:Npn \@@_V #1 #2
2434 {
2435   \str_if_eq:nnTF { #2 } { [ ] }
2436   { \@@_make_preamble_V_i:w [ ] }
2437   { \@@_make_preamble_V_i:w [ ] { #2 } }
2438 }
2439 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2440 { \@@_make_preamble_V_ii:nn { #1 } }
2441 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2442 {
2443   \str_set:Nn \l_@@_vpos_col_str { p }
2444   \str_set:Nn \l_@@_hpos_col_str { j }
2445   \tl_set:Nn \l_tmpa_tl { #1 }
2446   \@@_keys_p_column:V \l_tmpa_tl
2447   \IfPackageLoadedTF { varwidth }
2448   { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2449   {
2450     \@@_error_or_warning:n { varwidth~not~loaded }
2451     \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2452   }
2453 }

```

For w and W

```

2454 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2455 \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.

```

2456 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2457 {
2458   \str_if_eq:nnTF { #3 } { s }
2459   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2460   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2461 }

```

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.

```

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

```

```

2480     }
2481     \int_gincr:N \c@jCol
2482     \@@_rec_preamble_after_col:n
2483 }

```

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

```

2484 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2485 {
2486   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2487   \tl_gclear:N \g_@@_pre_cell_tl
2488   \tl_gput_right:Nn \g_@@_array_preamble_tl
2489   {
2490     > {

```

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.

```

2491       \dim_set:Nn \l_@@_col_width_dim { #4 }
2492       \hbox_set:Nw \l_@@_cell_box
2493       \@@_cell_begin:w
2494       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2495     }
2496     c
2497     < {
2498       \@@_cell_end:
2499       \hbox_set_end:
2500       #1
2501       \@@_adjust_size_box:
2502       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2503     }
2504   }

```

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

```

2505     \int_gincr:N \c@jCol
2506     \@@_rec_preamble_after_col:n
2507 }

```

```

2508 \cs_new_protected:Npn \@@_special_W:
2509 {
2510   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2511   { \@@_warning:n { W~warning } }
2512 }

```

For S (of siunitx).

```

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

```

```

2532         \siunitx_cell_begin:w
2533     }
2534     c
2535     < { \siunitx_cell_end: \@@_cell_end: }
2536 }

```

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

```

2537     \int_gincr:N \c@jCol
2538     \@@_rec_preamble_after_col:n
2539 }
2540 { \@@_fatal:n { siunitx-not-loaded } }
2541 }

```

For (, [and \{.

```

2542 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2543 {
2544     \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.

```

2545     \int_if_zero:nTF \c@jCol
2546     {
2547         \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2548         {

```

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

```

2549             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2550             \tl_gset:Nn \g_@@_right_delim_tl { . }
2551             \@@_rec_preamble:n #2
2552         }
2553         {
2554             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2555             \@@_make_preamble_iv:nn { #1 } { #2 }
2556         }
2557     }
2558     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2559 }
2560 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2561 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2562 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2563 {
2564     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2565     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2566     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2567     {
2568         \@@_error:nn { delimiter~after~opening } { #2 }
2569         \@@_rec_preamble:n
2570     }
2571     { \@@_rec_preamble:n #2 }
2572 }

```

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

```

2573 \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

For the closing delimiters. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have an opening delimiter following and we also have to see whether we are at the end of the preamble because, in that case, our letter must be considered as the right delimiter of the environment if the environment is {NiceArray}).

```

2574 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2575 {
2576     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2577     \tl_if_in:nnTF { ) ] \} } { #2 }
2578     { \@@_make_preamble_v:nnn #1 #2 }

```

```

2579 {
2580   \tl_if_eq:nnTF { \stop } { #2 }
2581   {
2582     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2583     { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2584     {
2585       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2586       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2587       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2588       \@@_rec_preamble:n #2
2589     }
2590   }
2591   {
2592     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2593     { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2594     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2595     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2596     \@@_rec_preamble:n #2
2597   }
2598 }
2599 }
2600 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ] }
2601 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2602 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2603 {
2604   \tl_if_eq:nnTF { \stop } { #3 }
2605   {
2606     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2607     {
2608       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2609       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2610       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2611       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2612     }
2613     {
2614       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2615       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2616       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2617       \@@_error:nn { double-closing-delimiter } { #2 }
2618     }
2619   }
2620   {
2621     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2622     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2623     \@@_error:nn { double-closing-delimiter } { #2 }
2624     \@@_rec_preamble:n #3
2625   }
2626 }
2627 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2628 { \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_U...} when the key vlines is used. In fact, we have also to test whether there is, after the <{...}, a @{...}.

```

2629 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2630 {
2631   \str_if_eq:nnTF { #1 } { < }
2632   \@@_rec_preamble_after_col:i:n
2633   {
2634     \str_if_eq:nnTF { #1 } { @ }
2635     \@@_rec_preamble_after_col:ii:n

```

```

2636     {
2637       \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2638       {
2639         \tl_gput_right:Nn \g_@@_array_preamble_tl
2640         { ! { \skip_horizontal:N \arrayrulewidth } }
2641       }
2642       {
2643         \exp_args:NNe
2644         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2645         {
2646           \tl_gput_right:Nn \g_@@_array_preamble_tl
2647           { ! { \skip_horizontal:N \arrayrulewidth } }
2648         }
2649       }
2650       \@@_rec_preamble:n { #1 }
2651     }
2652   }
2653 }
2654 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2655 {
2656   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2657   \@@_rec_preamble_after_col:n
2658 }

```

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.

```

2659 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2660 {
2661   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2662   {
2663     \tl_gput_right:Nn \g_@@_array_preamble_tl
2664     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2665   }
2666   {
2667     \exp_args:NNe
2668     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2669     {
2670       \tl_gput_right:Nn \g_@@_array_preamble_tl
2671       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2672     }
2673     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2674   }
2675   \@@_rec_preamble:n
2676 }
2677 \cs_new:cpn { @@ _ * } #1 #2 #3
2678 {
2679   \tl_clear:N \l_tmpa_tl
2680   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2681   \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpa_tl
2682 }

```

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

```

2683 \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.

```

2684 \cs_new:Npn \@@_X #1 #2
2685 {
2686   \str_if_eq:nnTF { #2 } { [ ]

```

```

2687 { \@@_make_preamble_X:w [ ]
2688 { \@@_make_preamble_X:w [ ] #2 }
2689 }

2690 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2691 { \@@_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/Up-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`).

```

2692 \keys_define:nn { WithArrows / X-column }
2693 { 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.

```

2694 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2695 {

```

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).

```

2696 \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).

```

2697 \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`.

```

2698 \int_zero_new:N \l_@@_weight_int
2699 \int_set:Nn \l_@@_weight_int { 1 }
2700 \tl_set:Nn \l_tmpa_tl { #1 }
2701 \@@_keys_p_column:V \l_tmpa_tl
2702 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2703 \int_compare:nNnT \l_@@_weight_int < 0
2704 {
2705   \@@_error_or_warning:n { negative-weight }
2706   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2707 }
2708 \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).

```

2709 \bool_if:NTF \l_@@_X_columns_aux_bool
2710 {
2711   \exp_args:Nne
2712   \@@_make_preamble_ii_iv:nnn
2713   { \l_@@_weight_int \l_@@_X_columns_dim }
2714   { minipage }
2715   { \@@_no_update_width: }
2716 }
2717 {
2718   \tl_gput_right:Nn \g_@@_array_preamble_tl
2719   {
2720     > {
2721       \@@_cell_begin:w
2722       \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`.

```

2723 \NotEmpty

```

The following code will nullify the box of the cell.

```
2724 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2725 { \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`.

```
2726 \begin { minipage } { 5 cm } \arraybackslash
2727 }
2728 c
2729 < {
2730 \end { minipage }
2731 \@@_cell_end:
2732 }
2733 }
2734 \int_gincr:N \c@jCol
2735 \@@_rec_preamble_after_col:n
2736 }
2737 }

2738 \cs_new_protected:Npn \@@_no_update_width:
2739 {
2740 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2741 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2742 }
```

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

```
2743 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2744 {
2745 \seq_gput_right:Nx \g_@@_cols_vlism_seq
2746 { \int_eval:n { \c@jCol + 1 } }
2747 \tl_gput_right:Nx \g_@@_array_preamble_tl
2748 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2749 \@@_rec_preamble:n
2750 }
```

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.

```
2751 \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).

```
2752 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2753 { \@@_fatal:n { Preamble~forgotten } }
2754 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2755 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2756 \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).

```
2757 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2758 {
```


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`.

```

2759 \multispan { #1 }
2760 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2761 \begingroup
2762 \cs_set:Npn \@addamp
2763 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2764 \tl_gclear:N \g_@@_preamble_tl
2765 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2766 \exp_args:NV \mkpream \g_@@_preamble_tl
2767 \@addtopreamble \@empty
2768 \endgroup

```

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

```

2769 \int_compare:nNnT { #1 } > 1
2770 {
2771   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2772   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2773   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2774   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2775   {
2776     {
2777       \int_if_zero:nTF \c@jCol
2778       { \int_eval:n { \c@iRow + 1 } }
2779       { \int_use:N \c@iRow }
2780     }
2781     { \int_eval:n { \c@jCol + 1 } }
2782     {
2783       \int_if_zero:nTF \c@jCol
2784       { \int_eval:n { \c@iRow + 1 } }
2785       { \int_use:N \c@iRow }
2786     }
2787     { \int_eval:n { \c@jCol + #1 } }
2788     { } % for the name of the block
2789   }
2790 }

```

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

```

2791 \cs_set:Npn \@sharp { #3 }
2792 \@arstrut
2793 \@preamble
2794 \null

```

We add some lines.

```

2795 \int_gadd:Nn \c@jCol { #1 - 1 }
2796 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2797 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2798 \ignorespaces
2799 }

```

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`.

```

2800 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2801 {
2802   \str_case:nnF { #1 }
2803   {

```

```

2804     c { \@@_make_m_preamble_i:n #1 }
2805     l { \@@_make_m_preamble_i:n #1 }
2806     r { \@@_make_m_preamble_i:n #1 }
2807     > { \@@_make_m_preamble_ii:nn #1 }
2808     ! { \@@_make_m_preamble_ii:nn #1 }
2809     @ { \@@_make_m_preamble_ii:nn #1 }
2810     | { \@@_make_m_preamble_iii:n #1 }
2811     p { \@@_make_m_preamble_iv:nnn t #1 }
2812     m { \@@_make_m_preamble_iv:nnn c #1 }
2813     b { \@@_make_m_preamble_iv:nnn b #1 }
2814     w { \@@_make_m_preamble_v:nnnn { } #1 }
2815     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2816     \q_stop { }
2817   }
2818   {
2819     \cs_if_exist:cTF { NC @ find @ #1 }
2820     {
2821       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2822       \exp_last_unbraced:NV \@@_make_m_preamble:n \l_tmpa_tl
2823     }
2824     {
2825       \tl_if_eq:nnT { #1 } { S }
2826       { \@@_fatal:n { unknown-column-type-S } }
2827       { \@@_fatal:nn { unknown-column-type } { #1 } }
2828     }
2829   }
2830 }

```

For c, l and r

```

2831 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2832 {
2833   \tl_gput_right:Nn \g_@@_preamble_tl
2834   {
2835     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2836     #1
2837     < \@@_cell_end:
2838   }

```

We test for the presence of a <.

```

2839   \@@_make_m_preamble_x:n
2840 }

```

For >, ! and @

```

2841 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2842 {
2843   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2844   \@@_make_m_preamble:n
2845 }

```

For |

```

2846 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2847 {
2848   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2849   \@@_make_m_preamble:n
2850 }

```

For p, m and b

```

2851 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2852 {
2853   \tl_gput_right:Nn \g_@@_preamble_tl
2854   {
2855     > {
2856       \@@_cell_begin:w

```

```

2857         \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2858         \mode_leave_vertical:
2859         \arraybackslash
2860         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2861     }
2862     c
2863     < {
2864         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2865         \end { minipage }
2866         \@@_cell_end:
2867     }
2868 }

```

We test for the presence of a <.

```

2869     \@@_make_m_preamble_x:n
2870 }

```

For w and W

```

2871 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2872 {
2873     \tl_gput_right:Nn \g_@@_preamble_tl
2874     {
2875         > {
2876             \dim_set:Nn \l_@@_col_width_dim { #4 }
2877             \hbox_set:Nw \l_@@_cell_box
2878             \@@_cell_begin:w
2879             \str_set:Nn \l_@@_hpos_cell_str { #3 }
2880         }
2881         c
2882         < {
2883             \@@_cell_end:
2884             \hbox_set_end:
2885             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2886             #1
2887             \@@_adjust_size_box:
2888             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2889         }
2890     }

```

We test for the presence of a <.

```

2891     \@@_make_m_preamble_x:n
2892 }

```

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

```

2893 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2894 {
2895     \str_if_eq:nnTF { #1 } { < }
2896     \@@_make_m_preamble_ix:n
2897     { \@@_make_m_preamble:n { #1 } }
2898 }
2899 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2900 {
2901     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2902     \@@_make_m_preamble_x:n
2903 }

```

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).

```

2904 \cs_new_protected:Npn \@@_put_box_in_flow:
2905 {
2906     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }

```

```

2907 \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2908 \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2909 { \box_use_drop:N \l_tmpa_box }
2910 \@@_put_box_in_flow_i:
2911 }

```

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).

```

2912 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2913 {
2914   \pgfpicture
2915   \@@_qpoint:n { row - 1 }
2916   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2917   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2918   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2919   \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).

```

2920   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2921   {
2922     \int_set:Nn \l_tmpa_int
2923     {
2924       \str_range:Nnn
2925       \l_@@_baseline_tl
2926       6
2927       { \tl_count:V \l_@@_baseline_tl }
2928     }
2929     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2930   }
2931   {
2932     \str_case:VnF \l_@@_baseline_tl
2933     {
2934       { t } { \int_set:Nn \l_tmpa_int 1 }
2935       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2936     }
2937     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2938     \bool_lazy_or:nnT
2939     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2940     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2941     {
2942       \@@_error:n { bad~value~for~baseline }
2943       \int_set:Nn \l_tmpa_int 1
2944     }
2945     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2946   \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2947   }
2948   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2949   \endpgfpicture
2950   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2951   \box_use_drop:N \l_tmpa_box
2952 }

```

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).

```

2953 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2954 {

```

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.

```

2955 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2956 {
2957   \int_compare:nNnT \c@jCol > 1 % added 2023-08-13
2958   {
2959     \box_set_wd:Nn \l_@@_the_array_box
2960     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2961   }
2962 }

```

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

```

2963 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2964 \bool_if:NT \l_@@_caption_above_bool
2965 {
2966   \tl_if_empty:NF \l_@@_caption_tl
2967   {
2968     \bool_set_false:N \g_@@_caption_finished_bool
2969     \int_gzero:N \c@tabularnote
2970     \@@_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).

```

2971 \int_compare:nNnT \g_@@_notes_caption_int > 0
2972 {
2973   \tl_build_gput_right:Nx \g_@@_aux_tl
2974   {
2975     \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2976     { \int_use:N \g_@@_notes_caption_int }
2977   }
2978   \int_gzero:N \g_@@_notes_caption_int
2979 }
2980 }
2981 }

```

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

```

2982 \hbox
2983 {
2984   \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.

```

2985 \@@_create_extra_nodes:
2986 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2987 }

```

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

```

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

```

```

2997 \end { minipage }
2998 }

```

```

2999 \cs_new_protected:Npn \@@_insert_caption:
3000 {
3001   \tl_if_empty:NF \l_@@_caption_tl
3002   {
3003     \cs_if_exist:NTF \@capttype
3004     { \@@_insert_caption_i: }
3005     { \@@_error:n { caption-outside~float } }
3006   }
3007 }

```

```

3008 \cs_new_protected:Npn \@@_insert_caption_i:
3009 {
3010   \group_begin:

```

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

```

3011   \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.

```

3012   \IfPackageLoadedTF { floatrow }
3013   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3014   { }
3015   \tl_if_empty:NTF \l_@@_short_caption_tl
3016   { \caption }
3017   { \caption [ \l_@@_short_caption_tl ] }
3018   { \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.

```

3019   \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
3020   {
3021     \bool_gset_true:N \g_@@_caption_finished_bool
3022     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3023     \int_gzero:N \c@tabularnote
3024   }
3025   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3026   \group_end:
3027 }
3028 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3029 {
3030   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3031   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3032 }

```

```

3033 \cs_new_protected:Npn \@@_insert_tabularnotes:
3034 {
3035   \seq_concat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3036   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3037   \skip_vertical:N 0.65ex

```

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

```

3038   \group_begin:
3039   \l_@@_notes_code_before_tl
3040   \tl_if_empty:NF \g_@@_tabularnote_tl
3041   {

```

```

3042     \g_@@_tabularnote_tl \par
3043     \tl_gclear:N \g_@@_tabularnote_tl
3044 }

```

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.

```

3045     \int_compare:nNnT \c@tabularnote > 0
3046     {
3047         \bool_if:NTF \l_@@_notes_para_bool
3048         {
3049             \begin { tabularnotes* }
3050             \seq_map_inline:Nn \g_@@_notes_seq
3051             { \@@_one_tabularnote:nn ##1 }
3052             \strut
3053             \end { tabularnotes* }

```

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

```

3054         \par
3055     }
3056     {
3057         \tabularnotes
3058         \seq_map_inline:Nn \g_@@_notes_seq
3059         { \@@_one_tabularnote:nn ##1 }
3060         \strut
3061         \endtabularnotes
3062     }
3063 }
3064 \unskip
3065 \group_end:
3066 \bool_if:NT \l_@@_notes_bottomrule_bool
3067 {
3068     \IfPackageLoadedTF { booktabs }
3069     {

```

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

```

3070         \skip_vertical:N \aboverulesep

```

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

```

3071         { \CT@arc@ \hrule height \heavyrulewidth }
3072     }
3073     { \@@_error_or_warning:n { bottomrule-without~booktabs } }
3074 }
3075 \l_@@_notes_code_after_tl
3076 \seq_gclear:N \g_@@_notes_seq
3077 \seq_gclear:N \g_@@_notes_in_caption_seq
3078 \int_gzero:N \c@tabularnote
3079 }

```

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.

```

3080 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3081 {
3082     \tl_if_novalue:nTF { #1 }
3083     { \item }
3084     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3085 }

```

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`.

```

3086 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:

```

```

3087 {
3088   \pgfpicture
3089     \@@_qpoint:n { row - 1 }
3090     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3091     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3092     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3093   \endpgfpicture
3094   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3095   \int_if_zero:nT \l_@@_first_row_int
3096   {
3097     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3098     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3099   }
3100   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3101 }

```

Now, the general case.

```

3102 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3103 {

```

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

```

3104   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3105   { \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`.

```

3106   \pgfpicture
3107   \@@_qpoint:n { row - 1 }
3108   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3109   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3110   {
3111     \int_set:Nn \l_tmpa_int
3112     {
3113       \str_range:Nnn
3114         \l_@@_baseline_tl
3115         6
3116       { \tl_count:V \l_@@_baseline_tl }
3117     }
3118     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3119   }
3120   {
3121     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3122     \bool_lazy_or:nnT
3123       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3124       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3125     {
3126       \@@_error:n { bad-value-for-baseline }
3127       \int_set:Nn \l_tmpa_int 1
3128     }
3129     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3130   }
3131   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3132   \endpgfpicture
3133   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3134   \int_if_zero:nT \l_@@_first_row_int
3135   {
3136     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3137     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3138   }
3139   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3140 }

```

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.

```

3141 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3142 {

```

We will compute the real width of both delimiters used.

```

3143 \dim_zero_new:N \l_@@_real_left_delim_dim
3144 \dim_zero_new:N \l_@@_real_right_delim_dim
3145 \hbox_set:Nn \l_tmpb_box
3146 {
3147   \c_math_toggle_token
3148   \left #1
3149   \vcenter
3150   {
3151     \vbox_to_ht:nn
3152     { \box_ht_plus_dp:N \l_tmpa_box }
3153     { }
3154   }
3155   \right .
3156   \c_math_toggle_token
3157 }
3158 \dim_set:Nn \l_@@_real_left_delim_dim
3159 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3160 \hbox_set:Nn \l_tmpb_box
3161 {
3162   \c_math_toggle_token
3163   \left .
3164   \vbox_to_ht:nn
3165   { \box_ht_plus_dp:N \l_tmpa_box }
3166   { }
3167   \right #2
3168   \c_math_toggle_token
3169 }
3170 \dim_set:Nn \l_@@_real_right_delim_dim
3171 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3172 \skip_horizontal:N \l_@@_left_delim_dim
3173 \skip_horizontal:N -\l_@@_real_left_delim_dim
3174 \@@_put_box_in_flow:
3175 \skip_horizontal:N \l_@@_right_delim_dim
3176 \skip_horizontal:N -\l_@@_real_right_delim_dim
3177 }

```

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).

```

3178 \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).

```

3179 {
3180   \peek_remove_spaces:n
3181   {
3182     \peek_meaning:NTF \end
3183     \@@_analyze_end:Nn
3184     {
3185       \@@_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`).

```

3186         \exp_args:NV \@@_array: \g_@@_array_preamble_tl
3187     }
3188 }
3189 }
3190 {
3191     \@@_create_col_nodes:
3192     \endarray
3193 }
```

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

```

3194 \NewDocumentEnvironment { @@-light-syntax } { b }
3195 {
```

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`.

```

3196     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
3197     \tl_map_inline:nn { #1 }
3198     {
3199         \str_if_eq:nnT { ##1 } { & }
3200         { \@@_fatal:n { ampersand~in~light-syntax } }
3201         \str_if_eq:nnT { ##1 } { \ }
3202         { \@@_fatal:n { double-backslash~in~light-syntax } }
3203     }
```

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`.

```

3204     \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```

3205 }
```

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.

```

3206 {
3207     \@@_create_col_nodes:
3208     \endarray
3209 }
3210 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3211 {
3212     \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).

```

3213     \seq_clear_new:N \l_@@_rows_seq
```

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

```

3214     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3215     \seq_set_split:NVN \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```

3216     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3217     \tl_if_empty:NF \l_tmpa_tl
3218     { \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.

```

3219   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3220     { \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`).

```

3221   \tl_build_begin:N \l_@@_new_body_tl
3222   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3223   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3224   \@@_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).

```

3225   \seq_map_inline:Nn \l_@@_rows_seq
3226   {
3227     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3228     \@@_line_with_light_syntax:n { ##1 }
3229   }
3230   \tl_build_end:N \l_@@_new_body_tl
3231   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3232   {
3233     \int_set:Nn \l_@@_last_col_int
3234     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3235   }

```

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.

```

3236   \@@_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`).

```

3237   \exp_args:NV \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3238 }

3239 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3240 {
3241   \seq_clear_new:N \l_@@_cells_seq
3242   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3243   \int_set:Nn \l_@@_nb_cols_int
3244   {
3245     \int_max:nn
3246     \l_@@_nb_cols_int
3247     { \seq_count:N \l_@@_cells_seq }
3248   }
3249   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3250   \exp_args:NNV \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3251   \seq_map_inline:Nn \l_@@_cells_seq
3252   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3253 }
3254 \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`.

```

3255 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3256 {
3257   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3258   { \@@_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.

```
3259     \end { #2 }
3260 }
```

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).

```
3261 \cs_new:Npn \@@_create_col_nodes:
3262 {
3263   \crrc
3264   \int_if_zero:nT \l_@@_first_col_int
3265   {
3266     \omit
3267     \hbox_overlap_left:n
3268     {
3269       \bool_if:NT \l_@@_code_before_bool
3270       { \pgfsys@markposition { \@@_env: - col - 0 } }
3271       \pgfpicture
3272       \pgfrememberpicturepositiononpagetrue
3273       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3274       \str_if_empty:NF \l_@@_name_str
3275       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3276       \endpgfpicture
3277       \skip_horizontal:N 2\col@sep
3278       \skip_horizontal:N \g_@@_width_first_col_dim
3279     }
3280     &
3281   }
3282   \omit
```

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

```
3283   \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`).

```
3284   \int_if_zero:nTF \l_@@_first_col_int
3285   {
3286     \bool_if:NT \l_@@_code_before_bool
3287     {
3288       \hbox
3289       {
3290         \skip_horizontal:N -0.5\arrayrulewidth
3291         \pgfsys@markposition { \@@_env: - col - 1 }
3292         \skip_horizontal:N 0.5\arrayrulewidth
3293       }
3294     }
3295     \pgfpicture
3296     \pgfrememberpicturepositiononpagetrue
3297     \pgfcoordinate { \@@_env: - col - 1 }
3298     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3299     \str_if_empty:NF \l_@@_name_str
3300     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3301     \endpgfpicture
3302   }
3303   {
3304     \bool_if:NT \l_@@_code_before_bool
3305     {
3306       \hbox
3307       {
3308         \skip_horizontal:N 0.5\arrayrulewidth
3309         \pgfsys@markposition { \@@_env: - col - 1 }
3310         \skip_horizontal:N -0.5\arrayrulewidth
```

```

3311     }
3312   }
3313   \pgfpicture
3314   \pgfrememberpicturepositiononpagetrue
3315   \pgfcoordinate { \l_@@_env: - col - 1 }
3316   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3317   \str_if_empty:NF \l_@@_name_str
3318   { \pgfnodealias { \l_@@_name_str - col - 1 } { \l_@@_env: - col - 1 } }
3319   \endpgfpicture
3320 }

```

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` (`0pt+plus 1fill`) but we will add some dimensions to it.

```

3321 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3322 \bool_if:NF \l_@@_auto_columns_width_bool
3323 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3324 {
3325   \bool_lazy_and:nnTF
3326   \l_@@_auto_columns_width_bool
3327   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3328   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3329   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3330   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3331 }
3332 \skip_horizontal:N \g_tmpa_skip
3333 \hbox
3334 {
3335   \bool_if:NT \l_@@_code_before_bool
3336   {
3337     \hbox
3338     {
3339       \skip_horizontal:N -0.5\arrayrulewidth
3340       \pgfsys@markposition { \l_@@_env: - col - 2 }
3341       \skip_horizontal:N 0.5\arrayrulewidth
3342     }
3343   }
3344   \pgfpicture
3345   \pgfrememberpicturepositiononpagetrue
3346   \pgfcoordinate { \l_@@_env: - col - 2 }
3347   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3348   \str_if_empty:NF \l_@@_name_str
3349   { \pgfnodealias { \l_@@_name_str - col - 2 } { \l_@@_env: - col - 2 } }
3350   \endpgfpicture
3351 }

```

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.

```

3352 \int_gset:Nn \g_tmpa_int 1
3353 \bool_if:NTF \g_@@_last_col_found_bool
3354 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3355 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3356 {
3357   &
3358   \omit
3359   \int_gincr:N \g_tmpa_int

```

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

```

3360   \skip_horizontal:N \g_tmpa_skip
3361   \bool_if:NT \l_@@_code_before_bool
3362   {

```

```

3363         \hbox
3364         {
3365             \skip_horizontal:N -0.5\arrayrulewidth
3366             \pgfsys@markposition
3367             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3368             \skip_horizontal:N 0.5\arrayrulewidth
3369         }
3370     }

```

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

```

3371     \pgfpicture
3372     \pgfrememberpicturepositiononpagetrue
3373     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3374     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3375     \str_if_empty:NF \l_@@_name_str
3376     {
3377         \pgfnodealias
3378         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3379         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3380     }
3381     \endpgfpicture
3382 }

```

```

3383     &
3384     \omit

```

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

```

3385     \int_compare:nNnT \g_@@_col_total_int = 1
3386     { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3387     \skip_horizontal:N \g_tmpa_skip
3388     \int_gincr:N \g_tmpa_int
3389     \bool_lazy_all:nT
3390     {
3391         { \bool_not_p:n \g_@@_delims_bool }
3392         { \bool_not_p:n \l_@@_tabular_bool }
3393         { \clist_if_empty_p:N \l_@@_vlines_clist }
3394         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3395         { ! \l_@@_bar_at_end_of_pream_bool }
3396     }
3397     { \skip_horizontal:N -\col@sep }
3398     \bool_if:NT \l_@@_code_before_bool
3399     {
3400         \hbox
3401         {
3402             \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.

```

3403             \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3404             { \skip_horizontal:N -\arraycolsep }
3405             \pgfsys@markposition
3406             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3407             \skip_horizontal:N 0.5\arrayrulewidth
3408             \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3409             { \skip_horizontal:N \arraycolsep }
3410         }
3411     }
3412     \pgfpicture
3413     \pgfrememberpicturepositiononpagetrue
3414     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3415     {
3416         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3417         {
3418             \pgfpoint
3419             { - 0.5 \arrayrulewidth - \arraycolsep }
3420             \c_zero_dim
3421         }
3422         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3423     }
3424     \str_if_empty:NF \l_@@_name_str
3425     {
3426         \pgfnodealias
3427         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3428         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3429     }
3430     \endpgfpicture

3431     \bool_if:NT \g_@@_last_col_found_bool
3432     {
3433         \hbox_overlap_right:n
3434         {
3435             \skip_horizontal:N \g_@@_width_last_col_dim
3436             \skip_horizontal:N \col@sep % added 2023-11-05
3437             \bool_if:NT \l_@@_code_before_bool
3438             {
3439                 \pgfsys@markposition
3440                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3441             }
3442             \pgfpicture
3443             \pgfrememberpicturepositiononpagetrue
3444             \pgfcoordinate
3445             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3446             \pgfpointorigin
3447             \str_if_empty:NF \l_@@_name_str
3448             {
3449                 \pgfnodealias
3450                 {
3451                     \l_@@_name_str - col
3452                     - \int_eval:n { \g_@@_col_total_int + 1 }
3453                 }
3454                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3455             }
3456             \endpgfpicture
3457         }
3458     }
3459     \cr
3460 }

```

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

```

3461 \tl_const:Nn \c_@@_preamble_first_col_tl
3462 {
3463     >
3464     {

```

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).

```

3465     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3466     \bool_gset_true:N \g_@@_after_col_zero_bool
3467     \@@_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.

```

3468     \hbox_set:Nw \l_@@_cell_box
3469     \@@_math_toggle_token:
3470     \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”.

```

3471     \bool_lazy_and:nnT
3472     { \int_compare_p:nNn \c@iRow > 0 }
3473     {
3474         \bool_lazy_or_p:nn
3475         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3476         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3477     }
3478     {
3479         \l_@@_code_for_first_col_tl
3480         \xglobal \colorlet { nicematrix-first-col } { . }
3481     }
3482 }

```

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`.

```

3483     l
3484     <
3485     {
3486         \@@_math_toggle_token:
3487         \hbox_set_end:
3488         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3489         \@@_adjust_size_box:
3490         \@@_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.

```

3491     \dim_gset:Nn \g_@@_width_first_col_dim
3492     { \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.

```

3493     \hbox_overlap_left:n
3494     {
3495         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3496         \@@_node_for_cell:
3497         { \box_use_drop:N \l_@@_cell_box }
3498         \skip_horizontal:N \l_@@_left_delim_dim
3499         \skip_horizontal:N \l_@@_left_margin_dim
3500         \skip_horizontal:N \l_@@_extra_left_margin_dim
3501     }
3502     \bool_gset_false:N \g_@@_empty_cell_bool
3503     \skip_horizontal:N -2\col@sep
3504 }
3505 }

```

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

```

3506 \tl_const:Nn \c_@@_preamble_last_col_tl
3507 {
3508     >
3509     {
3510         \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).

```

3511     \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.

```

3512     \bool_gset_true:N \g_@@_last_col_found_bool
3513     \int_gincr:N \c@jCol
3514     \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.

```

3515     \hbox_set:Nw \l_@@_cell_box
3516     \@@_math_toggle_token:

```



```
3517 \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”.

```
3518 \int_compare:nNnT \c@iRow > 0
3519 {
3520   \bool_lazy_or:nnT
3521   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3522   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3523   {
3524     \l_@@_code_for_last_col_tl
3525     \xglobal \colorlet { nicematrix-last-col } { . }
3526   }
3527 }
3528 }
3529 1
3530 <
3531 {
3532   \@@_math_toggle_token:
3533   \hbox_set_end:
3534   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3535   \@@_adjust_size_box:
3536   \@@_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.

```
3537 \dim_gset:Nn \g_@@_width_last_col_dim
3538 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3539 \skip_horizontal:N -2\col@sep
```

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

```
3540 \hbox_overlap_right:n
3541 {
3542   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3543   {
3544     \skip_horizontal:N \l_@@_right_delim_dim
3545     \skip_horizontal:N \l_@@_right_margin_dim
3546     \skip_horizontal:N \l_@@_extra_right_margin_dim
3547     \@@_node_for_cell:
3548   }
3549 }
3550 \bool_gset_false:N \g_@@_empty_cell_bool
3551 }
3552 }
```

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

```
3553 \NewDocumentEnvironment { NiceArray } { }
3554 {
3555   \bool_gset_false:N \g_@@_delims_bool
3556   \str_if_empty:NT \g_@@_name_env_str
3557   { \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).

```
3558 \NiceArrayWithDelims . .
3559 }
3560 { \endNiceArrayWithDelims }
```

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

```
3561 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3562 {
3563   \NewDocumentEnvironment { #1 NiceArray } { }
```

```

3564 {
3565   \bool_gset_true:N \g_@@_delims_bool
3566   \str_if_empty:NT \g_@@_name_env_str
3567   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3568   \@@_test_if_math_mode:
3569   \NiceArrayWithDelims #2 #3
3570 }
3571 { \endNiceArrayWithDelims }
3572 }
3573 \@@_def_env:nnn p ( )
3574 \@@_def_env:nnn b [ ]
3575 \@@_def_env:nnn B \{ \}
3576 \@@_def_env:nnn v | |
3577 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3578 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3579 {
3580   \bool_set_false:N \l_@@_preamble_bool
3581   \tl_clear:N \l_tmpa_tl
3582   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3583   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3584   \tl_put_right:Nn \l_tmpa_tl
3585   {
3586     *
3587     {
3588       \int_case:nnF \l_@@_last_col_int
3589       {
3590         { -2 } { \c@MaxMatrixCols }
3591         { -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).

```

3592   }
3593   { \int_eval:n { \l_@@_last_col_int - 1 } }
3594 }
3595 { #2 }
3596 }
3597 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3598 \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3599 }
3600 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3601 \clist_map_inline:nn { p , b , B , v , V }
3602 {
3603   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3604   {
3605     \bool_gset_true:N \g_@@_delims_bool
3606     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3607     % added 2023/10/01
3608     \int_if_zero:nT \l_@@_last_col_int
3609     {
3610       \bool_set_true:N \l_@@_last_col_without_value_bool
3611       \int_set:Nn \l_@@_last_col_int { -1 }
3612     }
3613     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3614     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3615   }
3616   { \use:c { end #1 NiceArray } }
3617 }

```

We define also an environment `{NiceMatrix}`

```

3618 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3619 {
3620   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3621   % added 2023/10/01
3622   \int_if_zero:nT \l_@@_last_col_int
3623   {
3624     \bool_set_true:N \l_@@_last_col_without_value_bool
3625     \int_set:Nn \l_@@_last_col_int { -1 }
3626   }
3627   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3628   \bool_lazy_or:nnT
3629     { \clist_if_empty_p:N \l_@@_vlines_clist }
3630     { \l_@@_except_borders_bool }
3631     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3632   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3633 }
3634 { \endNiceArray }

```

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

```

3635 \cs_new_protected:Npn \@@_NotEmpty:
3636 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3637 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3638 {

```

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`.

```

3639   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3640     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3641   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3642   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3643   \tl_if_empty:NF \l_@@_short_caption_tl
3644   {
3645     \tl_if_empty:NT \l_@@_caption_tl
3646     {
3647       \@@_error_or_warning:n { short-caption~without~caption }
3648       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3649     }
3650   }
3651   \tl_if_empty:NF \l_@@_label_tl
3652   {
3653     \tl_if_empty:NT \l_@@_caption_tl
3654     { \@@_error_or_warning:n { label~without~caption } }
3655   }
3656   \NewDocumentEnvironment { TabularNote } { b }
3657   {
3658     \bool_if:NTF \l_@@_in_code_after_bool
3659       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3660       {
3661         \tl_if_empty:NF \g_@@_tabularnote_tl
3662         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3663         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3664       }
3665   }
3666   { }
3667   \bool_set_true:N \l_@@_tabular_bool
3668   \NiceArray { #2 }
3669 }
3670 { \endNiceArray }

```

```

3671 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3672 {
3673   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3674   \dim_zero_new:N \l_@@_width_dim
3675   \dim_set:Nn \l_@@_width_dim { #1 }
3676   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3677   \bool_set_true:N \l_@@_tabular_bool
3678   \NiceArray { #3 }
3679 }
3680 {
3681   \endNiceArray
3682   \int_if_zero:nT \g_@@_total_X_weight_int
3683     { \@@_error:n { NiceTabularX~without~X } }
3684 }

3685 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3686 {
3687   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3688   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3689   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3690   \bool_set_true:N \l_@@_tabular_bool
3691   \NiceArray { #3 }
3692 }
3693 { \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`).

```

3694 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3695 {
3696   \bool_lazy_all:nT
3697     {
3698       { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3699       \l_@@_hvlines_bool
3700       { ! \g_@@_delims_bool }
3701       { ! \l_@@_except_borders_bool }
3702     }
3703     {
3704       \bool_set_true:N \l_@@_except_borders_bool
3705       \clist_if_empty:NF \l_@@_corners_clist
3706         { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3707       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3708         {
3709           \@@_stroke_block:nnn
3710           {
3711             rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3712             draw = \l_@@_rules_color_tl
3713           }
3714           { 1-1 }
3715           { \int_use:N \c@iRow - \int_use:N \c@jCol }
3716         }
3717     }
3718 }

3719 \cs_new_protected:Npn \@@_after_array:
3720 {
3721   \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.

```

3722 \bool_if:NT \g_@@_last_col_found_bool
3723 { \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`.

```

3724 \bool_if:NT \l_@@_last_col_without_value_bool
3725 { \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.

```

3726 \bool_if:NT \l_@@_last_row_without_value_bool
3727 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3728 \tl_build_gput_right:Nx \g_@@_aux_tl
3729 {
3730   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3731   {
3732     \int_use:N \l_@@_first_row_int ,
3733     \int_use:N \c@iRow ,
3734     \int_use:N \g_@@_row_total_int ,
3735     \int_use:N \l_@@_first_col_int ,
3736     \int_use:N \c@jCol ,
3737     \int_use:N \g_@@_col_total_int
3738   }
3739 }

```

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`).

```

3740 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3741 {
3742   \tl_build_gput_right:Nx \g_@@_aux_tl
3743   {
3744     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3745     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3746   }
3747 }
3748 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3749 {
3750   \tl_build_gput_right:Nx \g_@@_aux_tl
3751   {
3752     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3753     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3754     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3755     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3756   }
3757 }

```

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

```

3758 \@@_create_diag_nodes:

```

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

```

3759 \pgfpicture
3760 \int_step_inline:nn \c@iRow
3761 {
3762   \pgfnodealias
3763   { \@@_env: - ##1 - last }
3764   { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

3765     }
3766     \int_step_inline:nn \c@jCol
3767     {
3768         \pgfnodealias
3769         { \@@_env: - last - ##1 }
3770         { \@@_env: - \int_use:N \c@iRow - ##1 }
3771     }
3772     \str_if_empty:NF \l_@@_name_str
3773     {
3774         \int_step_inline:nn \c@iRow
3775         {
3776             \pgfnodealias
3777             { \l_@@_name_str - ##1 - last }
3778             { \@@_env: - ##1 - \int_use:N \c@jCol }
3779         }
3780         \int_step_inline:nn \c@jCol
3781         {
3782             \pgfnodealias
3783             { \l_@@_name_str - last - ##1 }
3784             { \@@_env: - \int_use:N \c@iRow - ##1 }
3785         }
3786     }
3787     \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.

```

3788     \bool_if:NT \l_@@_parallelize_diags_bool
3789     {
3790         \int_gzero_new:N \g_@@_ddots_int
3791         \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.

```

3792         \dim_gzero_new:N \g_@@_delta_x_one_dim
3793         \dim_gzero_new:N \g_@@_delta_y_one_dim
3794         \dim_gzero_new:N \g_@@_delta_x_two_dim
3795         \dim_gzero_new:N \g_@@_delta_y_two_dim
3796     }
3797     \int_zero_new:N \l_@@_initial_i_int
3798     \int_zero_new:N \l_@@_initial_j_int
3799     \int_zero_new:N \l_@@_final_i_int
3800     \int_zero_new:N \l_@@_final_j_int
3801     \bool_set_false:N \l_@@_initial_open_bool
3802     \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.

```

3803     \bool_if:NT \l_@@_small_bool
3804     {
3805         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3806         \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.

```

3807         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3808         { 0.6 \l_@@_xdots_shorten_start_dim }
3809         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

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

```

3810         { 0.6 \l_@@_xdots_shorten_end_dim }
3811     }

```

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

```

3812 \@@_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.

```

3813 \@@_compute_corners:

```

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

```

3814 \@@_adjust_pos_of_blocks_seq:
3815 \@@_deal_with_rounded_corners:
3816 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3817 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3818 \IfPackageLoadedTF { tikz }
3819 {
3820     \tikzset
3821     {
3822         every~picture / .style =
3823         {
3824             overlay ,
3825             remember~picture ,
3826             name~prefix = \@@_env: -
3827         }
3828     }
3829 }
3830 { }
3831 \cs_set_eq:NN \ialign \@@_old_ialign:
3832 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3833 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3834 \cs_set_eq:NN \OverBrace \@@_OverBrace
3835 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3836 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3837 \cs_set_eq:NN \line \@@_line
3838 \g_@@_pre_code_after_tl
3839 \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.

```

3840 \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).

```

3841 \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`).

```

3842 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3843 { \@@_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:`.

```

3844 \bool_set_true:N \l_@@_in_code_after_bool
3845 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3846 \scan_stop:
3847 \tl_gclear:N \g_nicematrix_code_after_tl
3848 \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.

```

3849 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3850 \tl_if_empty:NF \g_@@_pre_code_before_tl
3851 {
3852   \tl_build_gput_right:Nx \g_@@_aux_tl
3853   {
3854     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3855     { \exp_not:V \g_@@_pre_code_before_tl }
3856   }
3857   \tl_gclear:N \g_@@_pre_code_before_tl
3858 }
3859 \tl_if_empty:NF \g_nicematrix_code_before_tl
3860 {
3861   \tl_build_gput_right:Nx \g_@@_aux_tl
3862   {
3863     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3864     { \exp_not:V \g_nicematrix_code_before_tl }
3865   }
3866   \tl_gclear:N \g_nicematrix_code_before_tl
3867 }

3868 \str_gclear:N \g_@@_name_env_str
3869 \@@_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`.

```

3870 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3871 }

```

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`.

```

3872 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3873 { \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).

```

3874 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3875 {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`


```

3876 \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3877 { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3878 }

```

The following command must *not* be protected.

```

3879 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3880 {
3881   { #1 }
3882   { #2 }
3883   {
3884     \int_compare:nNnTF { #3 } > { 99 }
3885     { \int_use:N \c@iRow }
3886     { #3 }
3887   }
3888   {
3889     \int_compare:nNnTF { #4 } > { 99 }
3890     { \int_use:N \c@jCol }
3891     { #4 }
3892   }
3893   { #5 }
3894 }

```

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).

```

3895 \hook_gput_code:nnn { begindocument } { . }
3896 {
3897   \cs_new_protected:Npx \@@_draw_dotted_lines:
3898   {
3899     \c_@@_pgfortikzpicture_tl
3900     \@@_draw_dotted_lines_i:
3901     \c_@@_endpgfortikzpicture_tl
3902   }
3903 }

```

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

```

3904 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3905 {
3906   \pgfrememberpicturerepositiononpagetrue
3907   \pgf@relevantforpicturesizefalse
3908   \g_@@_HVdotsfor_lines_tl
3909   \g_@@_Vdots_lines_tl
3910   \g_@@_Ddots_lines_tl
3911   \g_@@_Iddots_lines_tl
3912   \g_@@_Cdots_lines_tl
3913   \g_@@_Ldots_lines_tl
3914 }

3915 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3916 {
3917   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3918   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3919 }

```

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

```

3920 \pgfdeclareshape { @@_diag_node }
3921 {
3922   \savedanchor { \five }
3923   {
3924     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

3925     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3926   }
3927   \anchor { 5 } { \five }
3928   \anchor { center } { \pgfpointorigin }
3929 }

```

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).

```

3930 \cs_new_protected:Npn \@@_create_diag_nodes:
3931 {
3932   \pgfpicture
3933   \pgfrememberpicturepositiononpagetrue
3934   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3935   {
3936     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3937     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3938     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3939     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3940     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3941     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3942     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3943     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3944     \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.

```

3945     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3946     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3947     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3948     \str_if_empty:NF \l_@@_name_str
3949     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3950   }

```

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

```

3951   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3952   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3953   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3954   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3955   \pgfcoordinate
3956   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3957   \pgfnodealias
3958   { \@@_env: - last }
3959   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3960   \str_if_empty:NF \l_@@_name_str
3961   {
3962     \pgfnodealias
3963     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3964     { \@@_env: - \int_use:N \l_tmpa_int }
3965     \pgfnodealias
3966     { \l_@@_name_str - last }
3967     { \@@_env: - last }
3968   }
3969   \endpgfpicture
3970 }

```

17 We draw the dotted lines

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

its right.

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

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

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

This command computes:

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

```
3971 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3972 {
```

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

```
3973   \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3974   \int_set:Nn \l_@@_initial_i_int { #1 }
3975   \int_set:Nn \l_@@_initial_j_int { #2 }
3976   \int_set:Nn \l_@@_final_i_int { #1 }
3977   \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.

```
3978   \bool_set_false:N \l_@@_stop_loop_bool
3979   \bool_do_until:Nn \l_@@_stop_loop_bool
3980   {
3981     \int_add:Nn \l_@@_final_i_int { #3 }
3982     \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3983     \bool_set_false:N \l_@@_final_open_bool
3984     \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3985     {
3986       \int_compare:nNnTF { #3 } = 1
3987       { \bool_set_true:N \l_@@_final_open_bool }
3988       {
3989         \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3990         { \bool_set_true:N \l_@@_final_open_bool }
3991       }
3992     }
3993   {
3994     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3995     {
3996       \int_compare:nNnTF { #4 } = { -1 }
3997       { \bool_set_true:N \l_@@_final_open_bool }
3998     }
3999   }
```

```

4000         \int_compare:nNtT \l_@@_final_j_int > \l_@@_col_max_int
4001         {
4002             \int_compare:nNtT { #4 } = 1
4003             { \bool_set_true:N \l_@@_final_open_bool }
4004         }
4005     }
4006 }
4007 \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.

```

4008     {

```

We do a step backwards.

```

4009         \int_sub:Nn \l_@@_final_i_int { #3 }
4010         \int_sub:Nn \l_@@_final_j_int { #4 }
4011         \bool_set_true:N \l_@@_stop_loop_bool
4012     }

```

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`.

```

4013     {
4014         \cs_if_exist:cTF
4015         {
4016             @@ _ dotted _
4017             \int_use:N \l_@@_final_i_int -
4018             \int_use:N \l_@@_final_j_int
4019         }
4020         {
4021             \int_sub:Nn \l_@@_final_i_int { #3 }
4022             \int_sub:Nn \l_@@_final_j_int { #4 }
4023             \bool_set_true:N \l_@@_final_open_bool
4024             \bool_set_true:N \l_@@_stop_loop_bool
4025         }
4026         {
4027             \cs_if_exist:cTF
4028             {
4029                 pgf @ sh @ ns @ \@@_env:
4030                 - \int_use:N \l_@@_final_i_int
4031                 - \int_use:N \l_@@_final_j_int
4032             }
4033             { \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.

```

4034         {
4035             \cs_set:cpn
4036             {
4037                 @@ _ dotted _
4038                 \int_use:N \l_@@_final_i_int -
4039                 \int_use:N \l_@@_final_j_int
4040             }
4041             { }
4042         }
4043     }
4044 }
4045 }

```

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

```

4046     \bool_set_false:N \l_@@_stop_loop_bool

```

```

4047 \bool_do_until:Nn \l_@@_stop_loop_bool
4048 {
4049   \int_sub:Nn \l_@@_initial_i_int { #3 }
4050   \int_sub:Nn \l_@@_initial_j_int { #4 }
4051   \bool_set_false:N \l_@@_initial_open_bool
4052   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4053   {
4054     \int_compare:nNnTF { #3 } = 1
4055     { \bool_set_true:N \l_@@_initial_open_bool }
4056     {
4057       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4058       { \bool_set_true:N \l_@@_initial_open_bool }
4059     }
4060   }
4061   {
4062     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4063     {
4064       \int_compare:nNnT { #4 } = 1
4065       { \bool_set_true:N \l_@@_initial_open_bool }
4066     }
4067     {
4068       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4069       {
4070         \int_compare:nNnT { #4 } = { -1 }
4071         { \bool_set_true:N \l_@@_initial_open_bool }
4072       }
4073     }
4074   }
4075   \bool_if:NTF \l_@@_initial_open_bool
4076   {
4077     \int_add:Nn \l_@@_initial_i_int { #3 }
4078     \int_add:Nn \l_@@_initial_j_int { #4 }
4079     \bool_set_true:N \l_@@_stop_loop_bool
4080   }
4081   {
4082     \cs_if_exist:cTF
4083     {
4084       @@ _ dotted _
4085       \int_use:N \l_@@_initial_i_int -
4086       \int_use:N \l_@@_initial_j_int
4087     }
4088     {
4089       \int_add:Nn \l_@@_initial_i_int { #3 }
4090       \int_add:Nn \l_@@_initial_j_int { #4 }
4091       \bool_set_true:N \l_@@_initial_open_bool
4092       \bool_set_true:N \l_@@_stop_loop_bool
4093     }
4094     {
4095       \cs_if_exist:cTF
4096       {
4097         pgf @ sh @ ns @ \@@_env:
4098         - \int_use:N \l_@@_initial_i_int
4099         - \int_use:N \l_@@_initial_j_int
4100       }
4101       { \bool_set_true:N \l_@@_stop_loop_bool }
4102       {
4103         \cs_set:cpn
4104         {
4105           @@ _ dotted _
4106           \int_use:N \l_@@_initial_i_int -
4107           \int_use:N \l_@@_initial_j_int
4108         }
4109         { }

```

```

4110         }
4111     }
4112 }
4113 }

```

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.

```

4114 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4115 {
4116     { \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`.

```

4117     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4118     { \int_use:N \l_@@_final_i_int }
4119     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4120     { } % for the name of the block
4121 }
4122 }

```

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.

```

4123 \cs_new_protected:Npn \@@_open_shorten:
4124 {
4125     \bool_if:NT \l_@@_initial_open_bool
4126     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4127     \bool_if:NT \l_@@_final_open_bool
4128     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4129 }

```

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).

```

4130 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4131 {
4132     \int_set:Nn \l_@@_row_min_int 1
4133     \int_set:Nn \l_@@_col_min_int 1
4134     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4135     \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`.

```

4136 \seq_map_inline:Nn \g_@@_submatrix_seq
4137 { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4138 }

```

`#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.

```

4139 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4140 {
4141     \bool_if:nT
4142     {
4143         \int_compare_p:n { #3 <= #1 }
4144         && \int_compare_p:n { #1 <= #5 }
4145         && \int_compare_p:n { #4 <= #2 }
4146         && \int_compare_p:n { #2 <= #6 }
4147     }

```

```

4148     {
4149         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4150         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4151         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4152         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4153     }
4154 }

4155 \cs_new_protected:Npn \@@_set_initial_coords:
4156 {
4157     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4158     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4159 }
4160 \cs_new_protected:Npn \@@_set_final_coords:
4161 {
4162     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4163     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4164 }
4165 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4166 {
4167     \pgfpointanchor
4168     {
4169         \@@_env:
4170         - \int_use:N \l_@@_initial_i_int
4171         - \int_use:N \l_@@_initial_j_int
4172     }
4173     { #1 }
4174     \@@_set_initial_coords:
4175 }
4176 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4177 {
4178     \pgfpointanchor
4179     {
4180         \@@_env:
4181         - \int_use:N \l_@@_final_i_int
4182         - \int_use:N \l_@@_final_j_int
4183     }
4184     { #1 }
4185     \@@_set_final_coords:
4186 }
4187 \cs_new_protected:Npn \@@_open_x_initial_dim:
4188 {
4189     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4190     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4191     {
4192         \cs_if_exist:cT
4193         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4194         {
4195             \pgfpointanchor
4196             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4197             { west }
4198             \dim_set:Nn \l_@@_x_initial_dim
4199             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4200         }
4201     }
4202 }
4203 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4204 {
4205     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4206     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4207     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4208 }

```

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

```

4209 \cs_new_protected:Npn \@@_open_x_final_dim:
4210 {
4211   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4212   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4213   {
4214     \cs_if_exist:cT
4215     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4216     {
4217       \pgfpointanchor
4218       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4219       { east }
4220       \dim_set:Nn \l_@@_x_final_dim
4221       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4222     }
4223   }

```

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

```

4224   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4225   {
4226     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4227     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4228     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4229   }
4230 }

```

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.

```

4231 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4232 {
4233   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4234   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4235   {
4236     \@@_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.

```

4237   \group_begin:
4238   \@@_open_shorten:
4239   \int_if_zero:nTF { #1 }
4240   { \color { nicematrix-first-row } }
4241   {

```

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.

```

4242     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4243     { \color { nicematrix-last-row } }
4244   }
4245   \keys_set:nn { NiceMatrix / xdots } { #3 }
4246   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4247   \@@_actually_draw_Ldots:
4248   \group_end:
4249 }
4250 }

```

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`.

```

4251 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4252 {
4253   \bool_if:NTF \l_@@_initial_open_bool
4254   {
4255     \@@_open_x_initial_dim:
4256     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4257     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4258   }
4259   { \@@_set_initial_coords_from_anchor:n { base-east } }
4260   \bool_if:NTF \l_@@_final_open_bool
4261   {
4262     \@@_open_x_final_dim:
4263     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4264     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4265   }
4266   { \@@_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.

```

4267   \bool_lazy_all:nTF
4268   {
4269     \l_@@_initial_open_bool
4270     \l_@@_final_open_bool
4271     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4272   }
4273   {
4274     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4275     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4276   }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of texte. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4277   {
4278     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4279     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4280   }
4281   \@@_draw_line:
4282 }

```

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.

```

4283 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4284 {
4285   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4286   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4287   {
4288     \@@_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.

```

4289   \group_begin:
4290   \@@_open_shorten:
4291   \int_if_zero:nTF { #1 }
4292   { \color { nicematrix-first-row } }
4293   {

```

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.

```

4294         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4295         { \color { nicematrix-last-row } }
4296     }
4297     \keys_set:nn { NiceMatrix / xdots } { #3 }
4298     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4299     \@@_actually_draw_Cdots:
4300 \group_end:
4301 }
4302 }
```

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`.

```

4303 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4304 {
4305     \bool_if:NTF \l_@@_initial_open_bool
4306     { \@@_open_x_initial_dim: }
4307     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4308     \bool_if:NTF \l_@@_final_open_bool
4309     { \@@_open_x_final_dim: }
4310     { \@@_set_final_coords_from_anchor:n { mid-west } }
4311     \bool_lazy_and:nnTF
4312     \l_@@_initial_open_bool
4313     \l_@@_final_open_bool
4314     {
4315         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4316         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4317         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4318         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4319         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4320     }
4321     {
4322         \bool_if:NT \l_@@_initial_open_bool
4323         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4324         \bool_if:NT \l_@@_final_open_bool
4325         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4326     }
4327     \@@_draw_line:
4328 }
4329 \cs_new_protected:Npn \@@_open_y_initial_dim:
4330 {
4331     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4332     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4333     {
4334         \cs_if_exist:cT
4335         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4336         {
4337             \pgfpointanchor
4338             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4339             { north }
4340             \dim_set:Nn \l_@@_y_initial_dim
```

```

4341         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4342     }
4343 }
4344 % modified 2023-08-10
4345 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4346 {
4347     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4348     \dim_set:Nn \l_@@_y_initial_dim
4349     {
4350         \fp_to_dim:n
4351         {
4352             \pgf@y
4353             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4354         }
4355     }
4356 }
4357 }
4358 \cs_new_protected:Npn \@@_open_y_final_dim:
4359 {
4360     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4361     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4362     {
4363         \cs_if_exist:cT
4364         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4365         {
4366             \pgfpointanchor
4367             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4368             { south }
4369             \dim_set:Nn \l_@@_y_final_dim
4370             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4371         }
4372     }
4373 % modified 2023-08-10
4374 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4375 {
4376     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4377     \dim_set:Nn \l_@@_y_final_dim
4378     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4379 }
4380 }

```

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.

```

4381 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4382 {
4383     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4384     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4385     {
4386         \@@_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.

```

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

```

```

4399     \group_end:
4400   }
4401 }

```

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`.

```

4402 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4403 {

```

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

```

4404   \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.

```

4405   {
4406     \@@_open_y_initial_dim:
4407     \@@_open_y_final_dim:
4408     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4409   {
4410     \@@_qpoint:n { col - 1 }
4411     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4412     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4413     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4414     % \bool_if:NT \g_@@_delims_bool
4415     % {
4416       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4417     % }
4418   }
4419   {
4420     \bool_lazy_and:nnTF
4421     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4422     { \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”.

```

4423   {
4424     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4425     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4426     \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4427     \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4428     % \bool_if:NT \g_@@_delims_bool
4429     % {
4430       \dim_add:Nn
4431       \l_@@_x_initial_dim
4432       \c_@@_shift_exterior_Vdots_dim
4433     % }
4434   }

```

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

```

4435   {
4436     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4437     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4438     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }

```

```

4439         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4440     }
4441 }
4442 }

```

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.

```

4443 {
4444     \bool_set_false:N \l_tmpa_bool
4445     \bool_lazy_and:nnT
4446     { ! \l_@@_initial_open_bool }
4447     { ! \l_@@_final_open_bool }
4448     {
4449         \@@_set_initial_coords_from_anchor:n { south-west }
4450         \@@_set_final_coords_from_anchor:n { north-west }
4451         \bool_set:Nn \l_tmpa_bool
4452         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4453     }

```

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

```

4454     \bool_if:NTF \l_@@_initial_open_bool
4455     {
4456         \@@_open_y_initial_dim:
4457         \@@_set_final_coords_from_anchor:n { north }
4458         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4459     }
4460     {
4461         \@@_set_initial_coords_from_anchor:n { south }
4462         \bool_if:NTF \l_@@_final_open_bool
4463         \@@_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.

```

4464     {
4465         \@@_set_final_coords_from_anchor:n { north }
4466         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4467         {
4468             \dim_set:Nn \l_@@_x_initial_dim
4469             {
4470                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4471                 \l_@@_x_initial_dim \l_@@_x_final_dim
4472             }
4473         }
4474     }
4475 }
4476 }
4477 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4478 \@@_draw_line:
4479 }

```

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.

```

4480 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4481 {
4482     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4483     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4484     {
4485         \@@_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.

```

4486     \group_begin:
4487     \@@_open_shorten:
4488     \keys_set:nn { NiceMatrix / xdots } { #3 }
4489     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4490     \@@_actually_draw_Ddots:
4491     \group_end:
4492   }
4493 }

```

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.`

```

4494 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4495 {
4496   \bool_if:NTF \l_@@_initial_open_bool
4497   {
4498     \@@_open_y_initial_dim:
4499     \@@_open_x_initial_dim:
4500   }
4501   { \@@_set_initial_coords_from_anchor:n { south-east } }
4502   \bool_if:NTF \l_@@_final_open_bool
4503   {
4504     \@@_open_x_final_dim:
4505     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4506   }
4507   { \@@_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.

```

4508   \bool_if:NT \l_@@_parallelize_diags_bool
4509   {
4510     \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).

```

4511     \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.

```

4512   {
4513     \dim_gset:Nn \g_@@_delta_x_one_dim
4514     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4515     \dim_gset:Nn \g_@@_delta_y_one_dim
4516     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4517   }

```

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`.

```

4518   {
4519     \dim_set:Nn \l_@@_y_final_dim
4520     {
4521       \l_@@_y_initial_dim +

```

```

4522         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4523         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4524     }
4525 }
4526 }
4527 \@@_draw_line:
4528 }

```

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.

```

4529 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4530 {
4531     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4532     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4533     {
4534         \@@_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.

```

4535     \group_begin:
4536     \@@_open_shorten:
4537     \keys_set:nn { NiceMatrix / xdots } { #3 }
4538     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4539     \@@_actually_draw_Iddots:
4540     \group_end:
4541 }
4542 }

```

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`.

```

4543 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4544 {
4545     \bool_if:NTF \l_@@_initial_open_bool
4546     {
4547         \@@_open_y_initial_dim:
4548         \@@_open_x_initial_dim:
4549     }
4550     { \@@_set_initial_coords_from_anchor:n { south-west } }
4551     \bool_if:NTF \l_@@_final_open_bool
4552     {
4553         \@@_open_y_final_dim:
4554         \@@_open_x_final_dim:
4555     }
4556     { \@@_set_final_coords_from_anchor:n { north-east } }
4557     \bool_if:NT \l_@@_parallelize_diags_bool
4558     {
4559         \int_gincr:N \g_@@_iddots_int
4560         \int_compare:nNnTF \g_@@_iddots_int = 1
4561         {
4562             \dim_gset:Nn \g_@@_delta_x_two_dim
4563             { \l_@@_x_final_dim - \l_@@_x_initial_dim }

```

```

4564         \dim_gset:Nn \g_@@_delta_y_two_dim
4565         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4566     }
4567     {
4568         \dim_set:Nn \l_@@_y_final_dim
4569         {
4570             \l_@@_y_initial_dim +
4571             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4572             \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4573         }
4574     }
4575 }
4576 \@@_draw_line:
4577 }

```

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`

```

4578 \cs_new_protected:Npn \@@_draw_line:
4579 {
4580     \pgfrememberpicturerepositiononpagetrue
4581     \pgf@relevantforpicturesizefalse
4582     \bool_lazy_or:nnTF
4583     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4584     \l_@@_dotted_bool
4585     \@@_draw_standard_dotted_line:
4586     \@@_draw_unstandard_dotted_line:
4587 }

```

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.

```

4588 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4589 {
4590     \begin { scope }
4591     \@@_draw_unstandard_dotted_line:o
4592     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4593 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4594 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4595 {
4596     \@@_draw_unstandard_dotted_line:nVWV
4597     { #1 }

```



```

4598     \l_@@_xdots_up_tl
4599     \l_@@_xdots_down_tl
4600     \l_@@_xdots_middle_tl
4601   }
4602   \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4603   \hook_gput_code:nnn { begindocument } { . }
4604   {
4605     \IfPackageLoadedTF { tikz }
4606     {
4607       \tikzset
4608       {
4609         @@_node_above / .style = { sloped , above } ,
4610         @@_node_below / .style = { sloped , below } ,
4611         @@_node_middle / .style =
4612         {
4613           sloped ,
4614           inner~sep = \c_@@_innersep_middle_dim
4615         }
4616       }
4617     }
4618     { }
4619   }

4620   \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4621   {

```

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.

```

4622   \dim_zero_new:N \l_@@_l_dim
4623   \dim_set:Nn \l_@@_l_dim
4624   {
4625     \fp_to_dim:n
4626     {
4627       sqrt
4628       (
4629         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4630         +
4631         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4632       )
4633     }
4634   }
4635   \bool_lazy_and:nnT % security
4636   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } < \c_@@_max_l_dim }
4637   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > { 1 pt } }
4638   {
4639     \dim_set:Nn \l_tmpa_dim
4640     {
4641       \l_@@_x_initial_dim
4642       + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4643       * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4644     }
4645     \dim_set:Nn \l_tmpb_dim
4646     {
4647       \l_@@_y_initial_dim
4648       + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4649       * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4650     }
4651     \dim_set:Nn \l_@@_tmpc_dim
4652     {
4653         \l_@@_x_final_dim
4654         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4655         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4656     }
4657     \dim_set:Nn \l_@@_tmpd_dim
4658     {
4659         \l_@@_y_final_dim
4660         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4661         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4662     }
4663     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4664     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4665     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4666     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4667 }

```

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

```

4668     \bool_if:NT \l_@@_xdots_h_labels_bool
4669     {
4670         \tikzset
4671         {
4672             @@_node_above / .style = { auto = left } ,
4673             @@_node_below / .style = { auto = right } ,
4674             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4675         }
4676     }
4677     \tl_if_empty:nF { #4 }
4678     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4679     \draw
4680     [ #1 ]
4681     ( \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).

```

4682     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4683     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4684     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4685     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4686     \end { scope }
4687 }
4688 \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).

```

4689 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4690 {
4691     \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.

```

4692     \dim_zero_new:N \l_@@_l_dim
4693     \dim_set:Nn \l_@@_l_dim
4694     {
4695         \fp_to_dim:n
4696         {
4697             sqrt
4698             (
4699                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4700                 +
4701                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4702             )

```

```

4703     }
4704 }

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.

4705     \bool_lazy_or:nnF
4706     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4707     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4708     \@@_draw_standard_dotted_line_i:
4709 \group_end:
4710 \bool_lazy_all:nF
4711 {
4712     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4713     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4714     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4715 }
4716 \l_@@_labels_standard_dotted_line:
4717 }
4718 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4719 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4720 {

```

The number of dots will be \l_tmpa_int+1 .

```

4721     \int_set:Nn \l_tmpa_int
4722     {
4723         \dim_ratio:nn
4724         {
4725             \l_@@_l_dim
4726             - \l_@@_xdots_shorten_start_dim
4727             - \l_@@_xdots_shorten_end_dim
4728         }
4729         \l_@@_xdots_inter_dim
4730     }

```

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

```

4731     \dim_set:Nn \l_tmpa_dim
4732     {
4733         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4734         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4735     }
4736     \dim_set:Nn \l_tmpb_dim
4737     {
4738         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4739         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4740     }

```

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.

```

4741     \dim_gadd:Nn \l_@@_x_initial_dim
4742     {
4743         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4744         \dim_ratio:nn
4745         {
4746             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4747             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4748         }
4749         { 2 \l_@@_l_dim }
4750     }
4751     \dim_gadd:Nn \l_@@_y_initial_dim

```

```

4752 {
4753   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4754   \dim_ratio:nn
4755   {
4756     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4757     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4758   }
4759   { 2 \l_@@_l_dim }
4760 }
4761 \pgf@relevantforpicturesizefalse
4762 \int_step_inline:nnn 0 \l_tmpa_int
4763 {
4764   \pgfpathcircle
4765   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4766   { \l_@@_xdots_radius_dim }
4767   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4768   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4769 }
4770 \pgfusepath{fill}
4771 }

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

```

```

4814 {
4815   \pgfnode
4816   { rectangle }
4817   { south }
4818   {
4819     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4820     {
4821       \c_math_toggle_token
4822       \scriptstyle \l_@@_xdots_up_tl
4823       \c_math_toggle_token
4824     }
4825   }
4826   { }
4827   { \pgfusepath { } }
4828 }
4829 \tl_if_empty:NF \l_@@_xdots_down_tl
4830 {
4831   \pgfnode
4832   { rectangle }
4833   { north }
4834   {
4835     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4836     {
4837       \c_math_toggle_token
4838       \scriptstyle \l_@@_xdots_down_tl
4839       \c_math_toggle_token
4840     }
4841   }
4842   { }
4843   { \pgfusepath { } }
4844 }
4845 \endpgfscope
4846 }

```

19 User commands available in the new environments

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

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

```

4847 \hook_gput_code:nnn { \begin{document} } { . }
4848 {
4849   \tl_set:Nn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4850   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4851   \cs_new_protected:Npn \@@_Ldots
4852     { \@@_collect_options:n { \@@_Ldots_i } }
4853   \exp_args:NNV \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4854     {
4855       \int_if_zero:nTF \c@jCol
4856       { \@@_error:nn { in-first-col } \Ldots }
4857       {
4858         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4859         { \@@_error:nn { in-last-col } \Ldots }
4860         {

```

```

4861         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4862         { #1 , down = #2 , up = #3 , middle = #4 }
4863     }
4864 }
4865 \bool_if:NF \l_@@_nullify_dots_bool
4866 { \phantom { \ensuremath { \@@_old_ldots } } }
4867 \bool_gset_true:N \g_@@_empty_cell_bool
4868 }

4869 \cs_new_protected:Npn \@@_Cdots
4870 { \@@_collect_options:n { \@@_Cdots_i } }
4871 \exp_args:NNV \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4872 {
4873     \int_if_zero:nTF \c@jCol
4874     { \@@_error:nn { in~first~col } \Cdots }
4875     {
4876         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4877         { \@@_error:nn { in~last~col } \Cdots }
4878         {
4879             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4880             { #1 , down = #2 , up = #3 , middle = #4 }
4881         }
4882     }
4883     \bool_if:NF \l_@@_nullify_dots_bool
4884     { \phantom { \ensuremath { \@@_old_cdots } } }
4885     \bool_gset_true:N \g_@@_empty_cell_bool
4886 }

4887 \cs_new_protected:Npn \@@_Vdots
4888 { \@@_collect_options:n { \@@_Vdots_i } }
4889 \exp_args:NNV \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4890 {
4891     \int_if_zero:nTF \c@iRow
4892     { \@@_error:nn { in~first~row } \Vdots }
4893     {
4894         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4895         { \@@_error:nn { in~last~row } \Vdots }
4896         {
4897             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4898             { #1 , down = #2 , up = #3 , middle = #4 }
4899         }
4900     }
4901     \bool_if:NF \l_@@_nullify_dots_bool
4902     { \phantom { \ensuremath { \@@_old_vdots } } }
4903     \bool_gset_true:N \g_@@_empty_cell_bool
4904 }

4905 \cs_new_protected:Npn \@@_Ddots
4906 { \@@_collect_options:n { \@@_Ddots_i } }
4907 \exp_args:NNV \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4908 {
4909     \int_case:nnF \c@iRow
4910     {
4911         0 { \@@_error:nn { in~first~row } \Ddots }
4912         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4913     }
4914     {
4915         \int_case:nnF \c@jCol
4916         {
4917             0 { \@@_error:nn { in~first~col } \Ddots }
4918             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }

```

```

4919     }
4920     {
4921         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4922         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4923         { #1 , down = #2 , up = #3 , middle = #4 }
4924     }
4925
4926     }
4927     \bool_if:NF \l_@@_nullify_dots_bool
4928     { \phantom { \ensuremath { \@@_old_ddots } } }
4929     \bool_gset_true:N \g_@@_empty_cell_bool
4930 }

```



```

4931 \cs_new_protected:Npn \@@_Iddots
4932 { \@@_collect_options:n { \@@_Iddots_i } }
4933 \exp_args:NNV \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4934 {
4935     \int_case:nnF \c@iRow
4936     {
4937         0 { \@@_error:nn { in~first~row } \Iddots }
4938         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4939     }
4940     {
4941         \int_case:nnF \c@jCol
4942         {
4943             0 { \@@_error:nn { in~first~col } \Iddots }
4944             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4945         }
4946         {
4947             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4948             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4949             { #1 , down = #2 , up = #3 , middle = #4 }
4950         }
4951     }
4952     \bool_if:NF \l_@@_nullify_dots_bool
4953     { \phantom { \ensuremath { \@@_old_iddots } } }
4954     \bool_gset_true:N \g_@@_empty_cell_bool
4955 }
4956 }

```

End of the \AddToHook.

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

```

4957 \keys_define:nn { NiceMatrix / Ddots }
4958 {
4959     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4960     draw-first .default:n = true ,
4961     draw-first .value_forbidden:n = true
4962 }

```

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

```

4963 \cs_new_protected:Npn \@@_Hspace:
4964 {
4965     \bool_gset_true:N \g_@@_empty_cell_bool
4966     \hspace
4967 }

```

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.

```

4968 \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`.

```

4969 \cs_new:Npn \@@_Hdotsfor:
4970 {
4971   \bool_lazy_and:nnTF
4972     { \int_if_zero_p:n \c@jCol }
4973     { \int_if_zero_p:n \l_@@_first_col_int }
4974     {
4975       \bool_if:NTF \g_@@_after_col_zero_bool
4976       {
4977         \multicolumn { 1 } { c } { }
4978         \@@_Hdotsfor_i
4979       }
4980       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4981     }
4982   {
4983     \multicolumn { 1 } { c } { }
4984     \@@_Hdotsfor_i
4985   }
4986 }

```

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:`).

```

4987 \hook_gput_code:nnn { begindocument } { . }
4988 {
4989   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4990   \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.

```

4991   \cs_new_protected:Npn \@@_Hdotsfor_i
4992     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
4993   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
4994     {
4995       \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4996       {
4997         \@@_Hdotsfor:nnnn
4998         { \int_use:N \c@iRow }
4999         { \int_use:N \c@jCol }
5000         { #2 }
5001         {
5002           #1 , #3 ,
5003           down = \exp_not:n { #4 } ,
5004           up = \exp_not:n { #5 } ,
5005           middle = \exp_not:n { #6 }
5006         }
5007       }
5008       \prg_replicate:nn { #2 - 1 }
5009       {
5010         &
5011         \multicolumn { 1 } { c } { }
5012         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i: % added 2023-08-26
5013       }
5014     }
5015 }

```

```

5016 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5017 {
5018   \bool_set_false:N \l_@@_initial_open_bool
5019   \bool_set_false:N \l_@@_final_open_bool

```


For the row, it's easy.

```
5020 \int_set:Nn \l_@@_initial_i_int { #1 }
5021 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5022 \int_compare:nNnTF { #2 } = 1
5023 {
5024   \int_set:Nn \l_@@_initial_j_int 1
5025   \bool_set_true:N \l_@@_initial_open_bool
5026 }
5027 {
5028   \cs_if_exist:cTF
5029   {
5030     pgf @ sh @ ns @ \@@_env:
5031     - \int_use:N \l_@@_initial_i_int
5032     - \int_eval:n { #2 - 1 }
5033   }
5034   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5035   {
5036     \int_set:Nn \l_@@_initial_j_int { #2 }
5037     \bool_set_true:N \l_@@_initial_open_bool
5038   }
5039 }
5040 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5041 {
5042   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5043   \bool_set_true:N \l_@@_final_open_bool
5044 }
5045 {
5046   \cs_if_exist:cTF
5047   {
5048     pgf @ sh @ ns @ \@@_env:
5049     - \int_use:N \l_@@_final_i_int
5050     - \int_eval:n { #2 + #3 }
5051   }
5052   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5053   {
5054     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5055     \bool_set_true:N \l_@@_final_open_bool
5056   }
5057 }
5058 \group_begin:
5059 \@@_open_shorten:
5060 \int_if_zero:nTF { #1 }
5061 { \color { nicematrix-first-row } }
5062 {
5063   \int_compare:nNnT { #1 } = \g_@@_row_total_int
5064   { \color { nicematrix-last-row } }
5065 }
5066
5067 \keys_set:nn { NiceMatrix / xdots } { #4 }
5068 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5069 \@@_actually_draw_Ldots:
5070 \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).

```
5071 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5072 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5073 }
```

```

5074 \hook_gput_code:nnn { begindocument } { . }
5075 {
5076   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5077   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5078   \cs_new_protected:Npn \@@_Vdotsfor:
5079     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5080   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5081     {
5082       \bool_gset_true:N \g_@@_empty_cell_bool
5083       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5084         {
5085           \@@_Vdotsfor:nnnn
5086           { \int_use:N \c@iRow }
5087           { \int_use:N \c@jCol }
5088           { #2 }
5089           {
5090             #1 , #3 ,
5091             down = \exp_not:n { #4 } ,
5092             up = \exp_not:n { #5 } ,
5093             middle = \exp_not:n { #6 }
5094           }
5095         }
5096     }
5097 }

5098 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5099 {
5100   \bool_set_false:N \l_@@_initial_open_bool
5101   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5102   \int_set:Nn \l_@@_initial_j_int { #2 }
5103   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5104   \int_compare:nNnTF { #1 } = 1
5105   {
5106     \int_set:Nn \l_@@_initial_i_int 1
5107     \bool_set_true:N \l_@@_initial_open_bool
5108   }
5109   {
5110     \cs_if_exist:cTF
5111     {
5112       pgf @ sh @ ns @ \@@_env:
5113       - \int_eval:n { #1 - 1 }
5114       - \int_use:N \l_@@_initial_j_int
5115     }
5116     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5117     {
5118       \int_set:Nn \l_@@_initial_i_int { #1 }
5119       \bool_set_true:N \l_@@_initial_open_bool
5120     }
5121   }
5122   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5123   {
5124     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5125     \bool_set_true:N \l_@@_final_open_bool
5126   }
5127   {
5128     \cs_if_exist:cTF
5129     {
5130       pgf @ sh @ ns @ \@@_env:
5131       - \int_eval:n { #1 + #3 }

```

```

5132         - \int_use:N \l_@@_final_j_int
5133     }
5134     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5135     {
5136         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5137         \bool_set_true:N \l_@@_final_open_bool
5138     }
5139 }

5140 \group_begin:
5141 \@@_open_shorten:
5142 \int_if_zero:nTF { #2 }
5143 { \color { nicematrix-first-col } }
5144 {
5145     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5146     { \color { nicematrix-last-col } }
5147 }
5148 \keys_set:nn { NiceMatrix / xdots } { #4 }
5149 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5150 \@@_actually_draw_Vdots:
5151 \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).

```

5152     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5153     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5154 }

```

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

```

5155 \NewDocumentCommand \@@_rotate: { 0 { } }
5156 {
5157     \peek_remove_spaces:n
5158     {
5159         \bool_gset_true:N \g_@@_rotate_bool
5160         \keys_set:nn { NiceMatrix / rotate } { #1 }
5161     }
5162 }

5163 \keys_define:nn { NiceMatrix / rotate }
5164 {
5165     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5166     c .value_forbidden:n = true ,
5167     unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
5168 }

```

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

```

5169 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5170 {
5171   \tl_if_empty:nTF { #2 }
5172     { #1 }
5173     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5174 }
5175 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5176 { \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).

```

5177 \hook_gput_code:nnn { begindocument } { . }
5178 {
5179   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5180   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5181   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
5182     {
5183       \group_begin:
5184       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5185       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5186       \use:e
5187       {
5188         \@@_line_i:nn
5189         { \@@_double_int_eval:n #2 - \q_stop }
5190         { \@@_double_int_eval:n #3 - \q_stop }
5191       }
5192       \group_end:
5193     }
5194 }
5195 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5196 {
5197   \bool_set_false:N \l_@@_initial_open_bool
5198   \bool_set_false:N \l_@@_final_open_bool
5199   \bool_if:nTF
5200     {
5201       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
5202       ||
5203       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
5204     }
5205     {
5206       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
5207     }

```

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

```

5208   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5209 }
5210 \hook_gput_code:nnn { begindocument } { . }
5211 {
5212   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5213   {

```

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:`.

```

5214   \c_@@_pgfortikzpicture_tl

```

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

```

5215     \@@_draw_line_iii:nn { #1 } { #2 }
5216     \c_@@_endpgfortikzpicture_tl
5217   }
5218 }

```

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

```

5219 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5220 {
5221   \pgfrememberpicturepositiononpagetrue
5222   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5223   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5224   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5225   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5226   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5227   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5228   \@@_draw_line:
5229 }

```

The commands \Ldots, \Cdots, \Vdots, \Ddots, and \Iddots don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

21 The command \RowStyle

\g_@@_row_style_tl may contain several instructions of the form:

```
\@@_if_row_less_than:nn_{\number_}\{instructions_}
```

Then, \g_@@_row_style_tl will be inserted in all the cells of the array (and also in both components of a \diagbox in a cell of in a mono-row block).

The test \@@_if_row_less_then:nn ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key **nb-rows** of \RowStyle).

That test will be active even in an expandable context because \@@_if_row_less_then:nn is *not* protected.

#1 is the first row *after* the scope of the instructions in **#2**

```

5230 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5231 { \int_compare:nNnT { \int_use:N \c@iRow } < { #1 } { #2 } }

```

\@@_put_in_row_style will be used several times by \RowStyle.

```

5232 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5233 {
5234   \tl_gput_right:Nx \g_@@_row_style_tl
5235   {

```

Be careful, \exp_not:N\@@_if_row_less_than:nn can't be replaced by a protected version of \@@_if_row_less_than:nn.

```

5236     \exp_not:N
5237     \@@_if_row_less_than:nn
5238     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5239     { \exp_not:n { #1 } }
5240   }
5241 }
5242 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5243 \keys_define:nn { NiceMatrix / RowStyle }
5244 {
5245   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5246   cell-space-top-limit .initial:n = \c_zero_dim ,
5247   cell-space-top-limit .value_required:n = true ,

```

```

5248 cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5249 cell-space-bottom-limit .initial:n = \c_zero_dim ,
5250 cell-space-bottom-limit .value_required:n = true ,
5251 cell-space-limits .meta:n =
5252 {
5253   cell-space-top-limit = #1 ,
5254   cell-space-bottom-limit = #1 ,
5255 } ,
5256 color .tl_set:N = \l_@@_color_tl ,
5257 color .value_required:n = true ,
5258 bold .bool_set:N = \l_tmpa_bool ,
5259 bold .default:n = true ,
5260 bold .initial:n = false ,
5261 nb-rows .code:n =
5262   \str_if_eq:nnTF { #1 } { * }
5263     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5264     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5265 nb-rows .value_required:n = true ,
5266 rowcolor .tl_set:N = \l_tmpa_tl ,
5267 rowcolor .value_required:n = true ,
5268 rowcolor .initial:n = ,
5269 unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5270 }

```

```

5271 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5272 {
5273   \group_begin:
5274   \tl_clear:N \l_tmpa_tl % value of \rowcolor
5275   \tl_clear:N \l_@@_color_tl
5276   \int_set:Nn \l_@@_key_nb_rows_int 1
5277   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5278   \tl_if_empty:NF \l_tmpa_tl
5279   {

```

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

```

5280     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5281     {

```

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

```

5282       \@@_exp_color_arg:N \@@_rectanglecolor \l_tmpa_tl
5283       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5284       { \int_use:N \c@iRow - * }
5285     }

```

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

```

5286     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
5287     {
5288       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5289       {
5290         \@@_exp_color_arg:N \@@_rowcolor \l_tmpa_tl
5291         {
5292           \int_eval:n { \c@iRow + 1 }
5293           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5294         }
5295       }
5296     }
5297   }
5298   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5299   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5300   {

```

```

5301     \@@_put_in_row_style:n
5302     {
5303         \exp_not:n
5304         {
5305             \tl_gput_right:Nn \g_@@_cell_after_hook_tl
5306             {
5307                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5308                 { \dim_use:N \l_tmpa_dim }
5309             }
5310         }
5311     }
5312 }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

5313     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5314     {
5315         \@@_put_in_row_style:n
5316         {
5317             \exp_not:n
5318             {
5319                 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
5320                 {
5321                     \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5322                     { \dim_use:N \l_tmpb_dim }
5323                 }
5324             }
5325         }
5326     }

```

\l_@@_color_tl is the value of the key color of \RowStyle.

```

5327     \tl_if_empty:NF \l_@@_color_tl
5328     {
5329         \@@_put_in_row_style:e
5330         {
5331             \mode_leave_vertical:
5332             \@@_color:n { \l_@@_color_tl }
5333         }
5334     }

```

\l_tmpa_bool is the value of the key bold.

```

5335     \bool_if:NT \l_tmpa_bool
5336     {
5337         \@@_put_in_row_style:n
5338         {
5339             \exp_not:n
5340             {
5341                 \if_mode_math:
5342                 \c_math_toggle_token
5343                 \bfseries \boldmath
5344                 \c_math_toggle_token
5345             }
5346             \else:
5347             \bfseries \boldmath
5348             \fi:
5349         }
5350     }
5351     \group_end:
5352     \g_@@_row_style_tl
5353     \ignorespaces
5354 }

```

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).

```
5355 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5356 {
```

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.

```
5357 \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`.

```
5358 \str_if_in:nnF { #1 } { !! }
5359 {
5360 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5361 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5362 }
5363 \int_if_zero:nTF \l_tmpa_int
```

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

```
5364 {
5365 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5366 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5367 }
```

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

```
5368 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5369 }
```

```
5370 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5371 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5372 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5373 {
5374 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5375 {
```


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

```

5376     \group_begin:
5377     \pgfsetcornersarced
5378     {
5379         \pgfpoint
5380         { \l_@@_tab_rounded_corners_dim }
5381         { \l_@@_tab_rounded_corners_dim }
5382     }

```

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.

```

5383     \bool_if:NTF \l_@@_hvlines_bool
5384     {
5385         \pgfpathrectanglecorners
5386         {
5387             \pgfpointadd
5388             { \@@_qpoint:n { row-1 } }
5389             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5390         }
5391         {
5392             \pgfpointadd
5393             {
5394                 \@@_qpoint:n
5395                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5396             }
5397             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5398         }
5399     }
5400     {
5401         \pgfpathrectanglecorners
5402         { \@@_qpoint:n { row-1 } }
5403         {
5404             \pgfpointadd
5405             {
5406                 \@@_qpoint:n
5407                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5408             }
5409             { \pgfpoint \c_zero_dim \arrayrulewidth }
5410         }
5411     }
5412     \pgfusepath { clip }
5413     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5414     }
5415 }

```

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`).

```

5416 \cs_new_protected:Npn \@@_actually_color:
5417 {
5418     \pgfpicture
5419     \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.

```

5420     \@@_clip_with_rounded_corners:
5421     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5422     {
5423         \begin { pgfscope }

```

```

5424         \@@_color_opacity ##2
5425         \use:c { g_@@_color _ ##1 _t1 }
5426         \tl_gclear:c { g_@@_color _ ##1 _t1 }
5427         \pgfusepath { fill }
5428     \end { pgfscope }
5429 }
5430 \endpgfpicture
5431 }

```

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

```

5432 \cs_new_protected:Npn \@@_color_opacity
5433 {
5434     \peek_meaning:NTF [
5435         { \@@_color_opacity:w }
5436         { \@@_color_opacity:w [ ] }
5437     }

```

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.

```

5438 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5439 {
5440     \tl_clear:N \l_tmpa_tl
5441     \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.

```

5442     \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5443     \tl_if_empty:NTF \l_tmpb_tl
5444         { \@declaredcolor }
5445         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5446 }

```

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

```

5447 \keys_define:nn { nicematrix / color-opacity }
5448 {
5449     opacity .tl_set:N          = \l_tmpa_tl ,
5450     opacity .value_required:n = true
5451 }

5452 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5453 {
5454     \tl_set:Nn \l_@@_rows_tl { #1 }
5455     \tl_set:Nn \l_@@_cols_tl { #2 }
5456     \@@_cartesian_path:
5457 }

```

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

```

5458 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5459 {
5460     \tl_if_blank:nF { #2 }
5461     {
5462         \@@_add_to_colors_seq:en
5463         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5464         { \@@_cartesian_color:nn { #3 } { - } }
5465     }
5466 }

```

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

```

5467 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5468 {
5469   \tl_if_blank:nF { #2 }
5470   {
5471     \@@_add_to_colors_seq:en
5472     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5473     { \@@_cartesian_color:nn { - } { #3 } }
5474   }
5475 }

```

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

```

5476 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5477 {
5478   \tl_if_blank:nF { #2 }
5479   {
5480     \@@_add_to_colors_seq:en
5481     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5482     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5483   }
5484 }

```

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

```

5485 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5486 {
5487   \tl_if_blank:nF { #2 }
5488   {
5489     \@@_add_to_colors_seq:en
5490     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5491     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5492   }
5493 }

```

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

```

5494 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5495 {
5496   \@@_cut_on_hyphen:w #1 \q_stop
5497   \tl_clear_new:N \l_@@_tmpc_tl
5498   \tl_clear_new:N \l_@@_tmpd_tl
5499   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5500   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5501   \@@_cut_on_hyphen:w #2 \q_stop
5502   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5503   \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`.

```

5504   \@@_cartesian_path:n { #3 }
5505 }

```

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

```

5506 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5507 {
5508   \clist_map_inline:nn { #3 }
5509   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5510 }

```

```

5511 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5512 {
5513   \int_step_inline:nn { \int_use:N \c@iRow }

```

```

5514 {
5515   \int_step_inline:nn { \int_use:N \c@jCol }
5516   {
5517     \int_if_even:nTF { ####1 + ##1 }
5518     { \@@_cellcolor [ #1 ] { #2 } }
5519     { \@@_cellcolor [ #1 ] { #3 } }
5520     { ##1 - ####1 }
5521   }
5522 }
5523 }

```

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”.

```

5524 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5525 {
5526   \@@_rectanglecolor [ #1 ] { #2 }
5527   { 1 - 1 }
5528   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5529 }

5530 \keys_define:nn { NiceMatrix / rowcolors }
5531 {
5532   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5533   respect-blocks .default:n = true ,
5534   cols .tl_set:N = \l_@@_cols_tl ,
5535   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5536   restart .default:n = true ,
5537   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5538 }

```

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*.

```

5539 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5540 {

```

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

```

5541   \group_begin:
5542   \seq_clear_new:N \l_@@_colors_seq
5543   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5544   \tl_clear_new:N \l_@@_cols_tl
5545   \tl_set:Nn \l_@@_cols_tl { - }
5546   \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).

```

5547   \int_zero_new:N \l_@@_color_int
5548   \int_set:Nn \l_@@_color_int 1
5549   \bool_if:NT \l_@@_respect_blocks_bool
5550   {

```

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`).

```

5551   \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5552   \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5553   { \@@_not_in_exterior_p:nnnnn ##1 }
5554 }

```

```

5555 \pgfpicture
5556 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5557 \clist_map_inline:nn { #2 }
5558 {
5559   \tl_set:Nn \l_tmpa_tl { ##1 }
5560   \tl_if_in:NnTF \l_tmpa_tl { - }
5561     { \@@_cut_on_hyphen:w ##1 \q_stop }
5562     { \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.

```

5563   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5564   \int_set:Nn \l_@@_color_int
5565     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5566   \int_zero_new:N \l_@@_tmpc_int
5567   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5568   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5569   {

```

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

```

5570     \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).

```

5571     \bool_if:NT \l_@@_respect_blocks_bool
5572     {
5573       \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5574       { \@@_intersect_our_row_p:nnnnn ###1 }
5575       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

5576     }
5577     \tl_set:Nx \l_@@_rows_tl
5578     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5579     \tl_clear_new:N \l_@@_color_tl
5580     \tl_set:Nx \l_@@_color_tl
5581     {
5582       \@@_color_index:n
5583       {
5584         \int_mod:nn
5585           { \l_@@_color_int - 1 }
5586           { \seq_count:N \l_@@_colors_seq }
5587         + 1
5588       }
5589     }
5590     \tl_if_empty:NF \l_@@_color_tl
5591     {
5592       \@@_add_to_colors_seq:ee
5593       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5594       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5595     }
5596     \int_incr:N \l_@@_color_int
5597     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5598   }
5599 }
5600 \endpgfpicture
5601 \group_end:
5602 }

```

The command `\@@_color_index:n` peeks in `\l_@@_colors_seq` the color at the index #1. However, if that color is the symbol `=`, the previous one is poken. This macro is recursive.

```

5603 \cs_new:Npn \@@_color_index:n #1
5604 {

```

```

5605 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5606 { \@@_color_index:n { #1 - 1 } }
5607 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5608 }

```

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.

```

5609 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5610 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

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

```

5611 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5612 {
5613   \int_compare:nNnT { #3 } > \l_tmpb_int
5614   { \int_set:Nn \l_tmpb_int { #3 } }
5615 }

5616 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5617 {
5618   \bool_lazy_or:nnTF
5619   { \int_if_zero_p:n { #4 } }
5620   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5621   \prg_return_false:
5622   \prg_return_true:
5623 }

```

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

```

5624 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5625 {
5626   \bool_if:nTF
5627   {
5628     \int_compare_p:n { #1 <= \l_tmpa_int }
5629     &&
5630     \int_compare_p:n { \l_tmpa_int <= #3 }
5631   }
5632   \prg_return_true:
5633   \prg_return_false:
5634 }

```

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`).

```

5635 \cs_new_protected:Npn \@@_cartesian_path:n #1
5636 {
5637   \bool_lazy_and:nnT
5638   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5639   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5640   {
5641     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5642     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5643   }

```

We begin the loop over the columns.

```

5644 \clist_map_inline:Nn \l_@@_cols_tl
5645 {
5646   \tl_set:Nn \l_tmpa_tl { ##1 }
5647   \tl_if_in:NnTF \l_tmpa_tl { - }
5648     { \@@_cut_on_hyphen:w ##1 \q_stop }
5649     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5650   \bool_lazy_or:nnT
5651     { \tl_if_blank_p:V \l_tmpa_tl }
5652     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5653     { \tl_set:Nn \l_tmpa_tl { 1 } }
5654   \bool_lazy_or:nnT
5655     { \tl_if_blank_p:V \l_tmpb_tl }
5656     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5657     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5658   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5659     { \tl_set:Nx \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5660   \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.

```

5661   \@@_qpoint:n { col - \l_tmpa_tl }
5662   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5663     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5664     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5665   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5666   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5667   \clist_map_inline:Nn \l_@@_rows_tl
5668   {
5669     \tl_set:Nn \l_tmpa_tl { #####1 }
5670     \tl_if_in:NnTF \l_tmpa_tl { - }
5671       { \@@_cut_on_hyphen:w #####1 \q_stop }
5672       { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5673     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5674     \tl_if_empty:NT \l_tmpb_tl
5675       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5676     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5677       { \tl_set:Nx \l_tmpb_tl { \int_use:N \g_@@_row_total_int } }

```

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

```

5678   \seq_if_in:NxF \l_@@_corners_cells_seq
5679   { \l_tmpa_tl - \l_@@_tmpc_tl }
5680   {
5681     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5682     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5683     \@@_qpoint:n { row - \l_tmpa_tl }
5684     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5685     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5686     \pgfpathrectanglecorners
5687       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5688       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5689   }
5690 }
5691 }
5692 }

```

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`).

```

5693 \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.

```

5694 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5695 {
5696   \clist_set_eq:NN \l_tmpa_clist #1
5697   \clist_clear:N #1
5698   \clist_map_inline:Nn \l_tmpa_clist
5699   {
5700     \tl_set:Nn \l_tmpa_tl { ##1 }
5701     \tl_if_in:NnTF \l_tmpa_tl { - }
5702     { \@@_cut_on_hyphen:w ##1 \q_stop }
5703     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5704     \bool_lazy_or:nnT
5705     { \tl_if_blank_p:V \l_tmpa_tl }
5706     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5707     { \tl_set:Nn \l_tmpa_tl { 1 } }
5708     \bool_lazy_or:nnT
5709     { \tl_if_blank_p:V \l_tmpb_tl }
5710     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5711     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5712     \int_compare:nNnT \l_tmpb_tl > #2
5713     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5714     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5715     { \clist_put_right:Nn #1 { ####1 } }
5716   }
5717 }

```

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

```

5718 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5719 {
5720   \@@_test_color_inside:
5721   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5722   {

```

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`).

```

5723     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5724     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5725   }
5726   \ignorespaces
5727 }

```

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

```

5728 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5729 {
5730   \@@_test_color_inside:
5731   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5732   {
5733     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5734     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5735     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5736   }
5737   \ignorespaces
5738 }

```

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).

```

5739 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5740 { \@@_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.

```

5741 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5742 {
5743   \@@_test_color_inside:
5744   \peek_remove_spaces:n
5745   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5746 }

5747 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5748 {

```

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`.

```

5749   \seq_gclear:N \g_tmpa_seq
5750   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5751   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5752   \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`.

```

5753   \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5754   {
5755     { \int_use:N \c@iRow }
5756     { \exp_not:n { #1 } }
5757     { \exp_not:n { #2 } }
5758     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5759   }
5760 }

```

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`).

```

5761 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5762 {
5763   \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).

```

5764   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5765   {
5766     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5767     {
5768       \@@_rowlistcolors
5769       [ \exp_not:n { #2 } ]
5770       { #1 - \int_eval:n { \c@iRow - 1 } }
5771       { \exp_not:n { #3 } }
5772       [ \exp_not:n { #4 } ]
5773     }
5774   }
5775 }

```

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.

```

5776 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5777 {
5778   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5779     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5780   \seq_gclear:N \g_@@_rowlistcolors_seq
5781 }

5782 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5783 {
5784   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5785     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5786 }

```

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.

```

5787 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5788 {

```

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).

```

5789   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5790   {

```

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).

```

5791     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5792     {
5793       \exp_not:N \columncolor [ #1 ]
5794       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5795     }
5796   }
5797 }

5798 \hook_gput_code:nnn { begindocument } { . }
5799 {
5800   \IfPackageLoadedTF { colortbl }
5801   {
5802     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5803     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5804     \cs_new_protected:Npn \@@_revert_colortbl:
5805     {
5806       \hook_gput_code:nnn { env / tabular / begin } { . }
5807       {
5808         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5809         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5810       }
5811     }
5812   }
5813   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5814 }

```

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`.

```
5815 \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`.

```
5816 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5817 {
5818   \int_if_zero:nTF \l_@@_first_col_int
5819     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5820     {
5821       \int_if_zero:nTF \c@jCol
5822         {
5823           \int_compare:nNf \c@iRow = { -1 }
5824             { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5825         }
5826       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5827     }
5828 }
```

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.

```
5829 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5830 {
5831   \int_if_zero:nF \c@iRow
5832   {
5833     \int_compare:nNf \c@iRow = \l_@@_last_row_int
5834     {
5835       \int_compare:nNt \c@jCol > \c_zero_int
5836       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5837     }
5838   }
5839 }
```

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:nNt\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.

```
5840 \keys_define:nn { NiceMatrix / Rules }
5841 {
5842   position .int_set:N = \l_@@_position_int ,
5843   position .value_required:n = true ,
5844   start .int_set:N = \l_@@_start_int ,
```

```

5845   start .initial:n = 1 ,
5846   end .code:n =
5847     \bool_lazy_or:nnTF
5848     { \tl_if_empty_p:n { #1 } }
5849     { \str_if_eq_p:nn { #1 } { last } }
5850     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5851     { \int_set:Nn \l_@@_end_int { #1 } }
5852   }

```

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.

```

5853 \keys_define:nn { NiceMatrix / RulesBis }
5854 {
5855   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5856   multiplicity .initial:n = 1 ,
5857   dotted .bool_set:N = \l_@@_dotted_bool ,
5858   dotted .initial:n = false ,
5859   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]`.

```

5860   color .code:n =
5861     \@@_set_CT@arc@:n { #1 }
5862     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5863   color .value_required:n = true ,
5864   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5865   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.

```

5866   tikz .code:n =
5867     \IfPackageLoadedTF { tikz }
5868     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5869     { \@@_error:n { tikz-without~tikz } } ,
5870   tikz .value_required:n = true ,
5871   total-width .dim_set:N = \l_@@_rule_width_dim ,
5872   total-width .value_required:n = true ,
5873   width .meta:n = { total-width = #1 } ,
5874   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5875 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5876 \cs_new_protected:Npn \@@_vline:n #1
5877 {

```

The group is for the options.

```

5878   \group_begin:
5879   \int_zero_new:N \l_@@_end_int
5880   \int_set_eq:NN \l_@@_end_int \c@iRow
5881   \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).

```

5882 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5883 \@@_vline_i:
5884 \group_end:
5885 }

```

```

5886 \cs_new_protected:Npn \@@_vline_i:
5887 {
5888 \int_zero_new:N \l_@@_local_start_int
5889 \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`.

```

5890 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5891 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5892 \l_tmpa_tl
5893 {

```

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.

```

5894 \bool_gset_true:N \g_tmpa_bool
5895 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5896 { \@@_test_vline_in_block:nnnnn ##1 }
5897 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5898 { \@@_test_vline_in_block:nnnnn ##1 }
5899 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5900 { \@@_test_vline_in_stroken_block:nnnn ##1 }
5901 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5902 \bool_if:NTF \g_tmpa_bool
5903 {
5904 \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.

```

5905 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5906 }
5907 {
5908 \int_compare:nNnT \l_@@_local_start_int > 0
5909 {
5910 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5911 \@@_vline_ii:
5912 \int_zero:N \l_@@_local_start_int
5913 }
5914 }
5915 }
5916 \int_compare:nNnT \l_@@_local_start_int > 0
5917 {
5918 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5919 \@@_vline_ii:
5920 }
5921 }

```

```

5922 \cs_new_protected:Npn \@@_test_in_corner_v:
5923 {
5924 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5925 {
5926 \seq_if_in:NxT
5927 \l_@@_corners_cells_seq
5928 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5929 { \bool_set_false:N \g_tmpa_bool }
5930 }

```

```

5931 {
5932   \seq_if_in:NxT
5933   \l_@@_corners_cells_seq
5934   { \l_tmpa_tl - \l_tmpb_tl }
5935   {
5936     \int_compare:nNnTF \l_tmpb_tl = 1
5937     { \bool_set_false:N \g_tmpa_bool }
5938     {
5939       \seq_if_in:NxT
5940       \l_@@_corners_cells_seq
5941       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5942       { \bool_set_false:N \g_tmpa_bool }
5943     }
5944   }
5945 }
5946 }

```

```

5947 \cs_new_protected:Npn \@@_vline_ii:
5948 {
5949   \tl_clear:N \l_@@_tikz_rule_tl
5950   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5951   \bool_if:NTF \l_@@_dotted_bool
5952   \@@_vline_iv:
5953   {
5954     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5955     \@@_vline_iii:
5956     \@@_vline_v:
5957   }
5958 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

5959 \cs_new_protected:Npn \@@_vline_iii:
5960 {
5961   \pgfpicture
5962   \pgfrememberpicturerepositiononpagetrue
5963   \pgf@relevantforpicturesizefalse
5964   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5965   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5966   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5967   \dim_set:Nn \l_tmpb_dim
5968   {
5969     \pgf@x
5970     - 0.5 \l_@@_rule_width_dim
5971     +
5972     ( \arrayrulewidth * \l_@@_multiplicity_int
5973       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5974   }
5975   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5976   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5977   \bool_lazy_all:nT
5978   {
5979     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5980     { \cs_if_exist_p:N \CT@drsc@ }
5981     { ! \tl_if_blank_p:V \CT@drsc@ }
5982   }
5983   {
5984     \group_begin:
5985     \CT@drsc@
5986     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5987     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5988     \dim_set:Nn \l_@@_tmpd_dim
5989     {

```

```

5990         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5991         * ( \l_@@_multiplicity_int - 1 )
5992     }
5993     \pgfpathrectanglecorners
5994     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5995     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5996     \pgfusepath { fill }
5997     \group_end:
5998 }
5999 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6000 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6001 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6002 {
6003     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6004     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6005     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6006     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6007 }
6008 \CT@arc@
6009 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6010 \pgfsetrectcap
6011 \pgfusepathqstroke
6012 \endpgfpicture
6013 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6014 \cs_new_protected:Npn \l_@@_vline_iv:
6015 {
6016     \pgfpicture
6017     \pgfrememberpicturepositiononpagetrue
6018     \pgf@relevantforpicturesizefalse
6019     \l_@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6020     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6021     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6022     \l_@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6023     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6024     \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6025     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6026     \CT@arc@
6027     \l_@@_draw_line:
6028     \endpgfpicture
6029 }

```

The following code is for the case when the user uses the key `tikz`.

```

6030 \cs_new_protected:Npn \l_@@_vline_v:
6031 {
6032     \begin {tikzpicture }
6033     % 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).

```

6034     \CT@arc@
6035     \tl_if_empty:NF \l_@@_rule_color_tl
6036     { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6037     \pgfrememberpicturepositiononpagetrue
6038     \pgf@relevantforpicturesizefalse
6039     \l_@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6040     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6041     \l_@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6042     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6043     \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6044     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y

```

```

6045 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6046 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6047 ( \l_tmpb_dim , \l_tmpa_dim ) --
6048 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6049 \end { tikzpicture }
6050 }

```

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).

```

6051 \cs_new_protected:Npn \@@_draw_vlines:
6052 {
6053   \int_step_inline:nnn
6054   {
6055     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6056     1 2
6057   }
6058   {
6059     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6060     { \int_eval:n { \c@jCol + 1 } }
6061     \c@jCol
6062   }
6063   {
6064     \tl_if_eq:NnF \l_@@_vlines_clist { all }
6065     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6066     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6067   }
6068 }

```

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}`.

```

6069 \cs_new_protected:Npn \@@_hline:n #1
6070 {

```

The group is for the options.

```

6071   \group_begin:
6072   \int_zero_new:N \l_@@_end_int
6073   \int_set_eq:NN \l_@@_end_int \c@jCol
6074   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6075   \@@_hline_i:
6076   \group_end:
6077 }

```

```

6078 \cs_new_protected:Npn \@@_hline_i:
6079 {
6080   \int_zero_new:N \l_@@_local_start_int
6081   \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`.

```

6082   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
6083   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6084   \l_tmpb_tl
6085   {

```

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.

```

6086     \bool_gset_true:N \g_tmpa_bool
6087     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6088     { \@@_test_hline_in_block:nnnnn ##1 }

```



```

6089 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6090 { \@@_test_hline_in_block:nnnnn ##1 }
6091 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6092 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6093 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6094 \bool_if:NTF \g_tmpa_bool
6095 {
6096   \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.

```

6097     { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6098   }
6099   {
6100     \int_compare:nNnT \l_@@_local_start_int > 0
6101     {
6102       \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6103       \@@_hline_ii:
6104       \int_zero:N \l_@@_local_start_int
6105     }
6106   }
6107 }
6108 \int_compare:nNnT \l_@@_local_start_int > 0
6109 {
6110   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6111   \@@_hline_ii:
6112 }
6113 }

```

```

6114 \cs_new_protected:Npn \@@_test_in_corner_h:
6115 {
6116   \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6117   {
6118     \seq_if_in:NxT
6119     \l_@@_corners_cells_seq
6120     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6121     { \bool_set_false:N \g_tmpa_bool }
6122   }
6123   {
6124     \seq_if_in:NxT
6125     \l_@@_corners_cells_seq
6126     { \l_tmpa_tl - \l_tmpb_tl }
6127     {
6128       \int_compare:nNnTF \l_tmpa_tl = 1
6129       { \bool_set_false:N \g_tmpa_bool }
6130       {
6131         \seq_if_in:NxT
6132         \l_@@_corners_cells_seq
6133         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6134         { \bool_set_false:N \g_tmpa_bool }
6135       }
6136     }
6137   }
6138 }

```

```

6139 \cs_new_protected:Npn \@@_hline_ii:
6140 {
6141   \tl_clear:N \l_@@_tikz_rule_tl
6142   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6143   \bool_if:NTF \l_@@_dotted_bool
6144   \@@_hline_iv:
6145   {

```

```

6146     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6147     \@@_hline_iii:
6148     \@@_hline_v:
6149   }
6150 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6151 \cs_new_protected:Npn \@@_hline_iii:
6152 {
6153   \pgfpicture
6154   \pgfrememberpicturepositiononpagetrue
6155   \pgf@relevantforpicturesizefalse
6156   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6157   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6158   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6159   \dim_set:Nn \l_tmpb_dim
6160   {
6161     \pgf@y
6162     - 0.5 \l_@@_rule_width_dim
6163     +
6164     ( \arrayrulewidth * \l_@@_multiplicity_int
6165       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6166   }
6167   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6168   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6169   \bool_lazy_all:nT
6170   {
6171     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6172     { \cs_if_exist_p:N \CT@drsc@ }
6173     { ! \tl_if_blank_p:V \CT@drsc@ }
6174   }
6175   {
6176     \group_begin:
6177     \CT@drsc@
6178     \dim_set:Nn \l_@@_tmpd_dim
6179     {
6180       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6181       * ( \l_@@_multiplicity_int - 1 )
6182     }
6183     \pgfpathrectanglecorners
6184     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6185     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6186     \pgfusepathqfill
6187     \group_end:
6188   }
6189   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6190   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6191   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6192   {
6193     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6194     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6195     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6196     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6197   }
6198   \CT@arc@
6199   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6200   \pgfsetrectcap
6201   \pgfusepathqstroke
6202   \endpgfpicture
6203 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6204 \cs_new_protected:Npn \@@_hline_iv:
6205 {
6206   \pgfpicture
6207   \pgfrememberpicturepositiononpagetrue
6208   \pgf@relevantforpicturesizefalse
6209   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6210   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6211   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6212   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6213   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6214   \int_compare:nNnT \l_@@_local_start_int = 1
6215   {
6216     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6217     \bool_if:NF \g_@@_delims_bool
6218     { \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.

```

6219   \tl_if_eq:NnF \g_@@_left_delim_tl (
6220     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6221   )
6222   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6223   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6224   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6225   {
6226     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6227     \bool_if:NF \g_@@_delims_bool
6228     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6229     \tl_if_eq:NnF \g_@@_right_delim_tl )
6230     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6231   }
6232   \CT@arc@
6233   \@@_draw_line:
6234   \endpgfpicture
6235 }

```

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`).

```

6236 \cs_new_protected:Npn \@@_hline_v:
6237 {
6238   \begin { tikzpicture }
6239   % 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).

```

6240   \CT@arc@

```

```

6241 \tl_if_empty:NF \l_@@_rule_color_tl
6242 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6243 \pgfrememberpicturepositiononpagetrue
6244 \pgf@relevantforpicturesizefalse
6245 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6246 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6247 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6248 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6249 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6250 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6251 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6252 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6253 ( \l_tmpa_dim , \l_tmpb_dim ) --
6254 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6255 \end { tikzpicture }
6256 }

```

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).

```

6257 \cs_new_protected:Npn \@@_draw_hlines:
6258 {
6259   \int_step_inline:nnn
6260   {
6261     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6262     1 2
6263   }
6264   {
6265     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6266     { \int_eval:n { \c@iRow + 1 } }
6267     \c@iRow
6268   }
6269   {
6270     \tl_if_eq:NnF \l_@@_hlines_clist { all }
6271     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6272     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6273   }
6274 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6275 \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.

```

6276 \cs_set:Npn \@@_Hline_i:n #1
6277 {
6278   \peek_remove_spaces:n
6279   {
6280     \peek_meaning:NTF \Hline
6281     { \@@_Hline_ii:nn { #1 + 1 } }
6282     { \@@_Hline_iii:n { #1 } }
6283   }
6284 }
6285 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6286 \cs_set:Npn \@@_Hline_iii:n #1
6287 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6288 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6289 {
6290   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6291   \skip_vertical:n { \l_@@_rule_width_dim }
6292   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6293   {

```

```

6294 \@@_hline:n
6295 {
6296     multiplicity = #1 ,
6297     position = \int_eval:n { \c@iRow + 1 } ,
6298     total-width = \dim_use:N \l_@@_rule_width_dim ,
6299     #2
6300 }
6301 }
6302 \egroup
6303 }

```

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`).

```

6304 \cs_new_protected:Npn \@@_custom_line:n #1
6305 {
6306     \str_clear_new:N \l_@@_command_str
6307     \str_clear_new:N \l_@@_ccommand_str
6308     \str_clear_new:N \l_@@_letter_str
6309     \tl_clear_new:N \l_@@_other_keys_tl
6310     \keys_set: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.

```

6311 \bool_lazy_all:nTF
6312 {
6313     { \str_if_empty_p:N \l_@@_letter_str }
6314     { \str_if_empty_p:N \l_@@_command_str }
6315     { \str_if_empty_p:N \l_@@_ccommand_str }
6316 }
6317 { \@@_error:n { No-letter~and~no-command } }
6318 { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
6319 }

6320 \keys_define:nn { NiceMatrix / custom-line }
6321 {
6322     letter .str_set:N = \l_@@_letter_str ,
6323     letter .value_required:n = true ,
6324     command .str_set:N = \l_@@_command_str ,
6325     command .value_required:n = true ,
6326     ccommand .str_set:N = \l_@@_ccommand_str ,
6327     ccommand .value_required:n = true ,
6328 }

```

```

6329 \cs_new_protected:Npn \@@_custom_line_i:n #1
6330 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6331 \bool_set_false:N \l_@@_tikz_rule_bool
6332 \bool_set_false:N \l_@@_dotted_rule_bool
6333 \bool_set_false:N \l_@@_color_bool

6334 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6335 \bool_if:NT \l_@@_tikz_rule_bool
6336 {
6337     \IfPackageLoadedTF { tikz }
6338     { }

```

```

6339         { \@@_error:n { tikz~in~custom~line~without~tikz } }
6340     \bool_if:NT \l_@@_color_bool
6341         { \@@_error:n { color~in~custom~line~with~tikz } }
6342     }
6343 \bool_if:nT
6344     {
6345         \int_compare_p:nNn \l_@@_multiplicity_int > 1
6346         && \l_@@_dotted_rule_bool
6347     }
6348     { \@@_error:n { key~multiplicity~with~dotted } }
6349 \str_if_empty:NF \l_@@_letter_str
6350     {
6351         \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6352         { \@@_error:n { Several~letters } }
6353         {
6354             \exp_args:NnV \tl_if_in:NnTF
6355             \c_@@_forbidden_letters_str \l_@@_letter_str
6356             { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6357             {

```

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.

```

6358         \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6359         { \@@_v_custom_line:n { #1 } }
6360     }
6361 }
6362 }
6363 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6364 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6365 }
6366 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6367 \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.

```

6368 \keys_define:nn { NiceMatrix / custom-line-bis }
6369 {
6370     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6371     multiplicity .initial:n = 1 ,
6372     multiplicity .value_required:n = true ,
6373     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6374     color .value_required:n = true ,
6375     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6376     tikz .value_required:n = true ,
6377     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6378     dotted .value_forbidden:n = true ,
6379     total-width .code:n = { } ,
6380     total-width .value_required:n = true ,
6381     width .code:n = { } ,
6382     width .value_required:n = true ,
6383     sep-color .code:n = { } ,
6384     sep-color .value_required:n = true ,
6385     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6386 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6387 \bool_new:N \l_@@_dotted_rule_bool
6388 \bool_new:N \l_@@_tikz_rule_bool
6389 \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`.

```

6390 \keys_define:nn { NiceMatrix / custom-line-width }
6391 {
6392   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6393   multiplicity .initial:n = 1 ,
6394   multiplicity .value_required:n = true ,
6395   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6396   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6397                       \bool_set_true:N \l_@@_total_width_bool ,
6398   total-width .value_required:n = true ,
6399   width .meta:n = { total-width = #1 } ,
6400   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6401 }

```

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`).

```

6402 \cs_new_protected:Npn \@@_h_custom_line:n #1
6403 {

```

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`).

```

6404   \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6405   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6406 }

```

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`).

```

6407 \cs_new_protected:Npn \@@_c_custom_line:n #1
6408 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6409   \exp_args:Nc \NewExpandableDocumentCommand
6410     { nicematrix - \l_@@_ccommand_str }
6411     { 0 { } m }
6412     {
6413       \noalign
6414       {
6415         \@@_compute_rule_width:n { #1 , ##1 }
6416         \skip_vertical:n { \l_@@_rule_width_dim }
6417         \clist_map_inline:nn
6418           { ##2 }
6419           { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6420       }
6421     }
6422   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6423 }

```

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*.

```

6424 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6425 {
6426   \str_if_in:nnTF { #2 } { - }
6427     { \@@_cut_on_hyphen:w #2 \q_stop }
6428     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6429   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6430   {
6431     \@@_hline:n
6432     {

```

```

6433         #1 ,
6434         start = \l_tmpa_tl ,
6435         end = \l_tmpb_tl ,
6436         position = \int_eval:n { \c@iRow + 1 } ,
6437         total-width = \dim_use:N \l_@@_rule_width_dim
6438     }
6439 }
6440 }

6441 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6442 {
6443     \bool_set_false:N \l_@@_tikz_rule_bool
6444     \bool_set_false:N \l_@@_total_width_bool
6445     \bool_set_false:N \l_@@_dotted_rule_bool
6446     \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6447     \bool_if:NF \l_@@_total_width_bool
6448     {
6449         \bool_if:NTF \l_@@_dotted_rule_bool
6450         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6451         {
6452             \bool_if:NF \l_@@_tikz_rule_bool
6453             {
6454                 \dim_set:Nn \l_@@_rule_width_dim
6455                 {
6456                     \arrayrulewidth * \l_@@_multiplicity_int
6457                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6458                 }
6459             }
6460         }
6461     }
6462 }

6463 \cs_new_protected:Npn \@@_v_custom_line:n #1
6464 {
6465     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6466     \tl_gput_right:Nx \g_@@_array_preamble_tl
6467     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6468     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6469     {
6470         \@@_vline:n
6471         {
6472             #1 ,
6473             position = \int_eval:n { \c@jCol + 1 } ,
6474             total-width = \dim_use:N \l_@@_rule_width_dim
6475         }
6476     }
6477     \@@_rec_preamble:n
6478 }

6479 \@@_custom_line:n
6480 { 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.

```

6481 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6482 {
6483     \bool_lazy_all:nT
6484     {
6485         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }

```



```

6486     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6487     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6488     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6489   }
6490   { \bool_gset_false:N \g_tmpa_bool }
6491 }

```

The same for vertical rules.

```

6492 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6493 {
6494   \bool_lazy_all:nT
6495   {
6496     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6497     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6498     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6499     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6500   }
6501   { \bool_gset_false:N \g_tmpa_bool }
6502 }
6503 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6504 {
6505   \bool_lazy_all:nT
6506   {
6507     {
6508       ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6509       || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6510     }
6511     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6512     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6513   }
6514   { \bool_gset_false:N \g_tmpa_bool }
6515 }
6516 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6517 {
6518   \bool_lazy_all:nT
6519   {
6520     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6521     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6522     {
6523       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6524       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6525     }
6526   }
6527   { \bool_gset_false:N \g_tmpa_bool }
6528 }

```

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.

```

6529 \cs_new_protected:Npn \@@_compute_corners:
6530 {

```

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.

```

6531   \seq_clear_new:N \l_@@_corners_cells_seq
6532   \clist_map_inline:Nn \l_@@_corners_clist

```

```

6533 {
6534   \str_case:nnF { ##1 }
6535   {
6536     { NW }
6537     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6538     { NE }
6539     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6540     { SW }
6541     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6542     { SE }
6543     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6544   }
6545   { \@@_error:nn { bad~corner } { ##1 } }
6546 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6547   \seq_if_empty:NF \l_@@_corners_cells_seq
6548   {

```

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.

```

6549   \tl_build_gput_right:Nx \g_@@_aux_tl
6550   {
6551     \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6552     { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6553   }
6554 }
6555 }

```

“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.

```

6556 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6557 {

```

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.

```

6558   \bool_set_false:N \l_tmpa_bool
6559   \int_zero_new:N \l_@@_last_empty_row_int
6560   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6561   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6562   {
6563     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6564     \bool_lazy_or:nnTF
6565     {
6566       \cs_if_exist_p:c
6567       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6568     }
6569     \l_tmpb_bool
6570     { \bool_set_true:N \l_tmpa_bool }
6571   }

```

```

6572         \bool_if:NF \l_tmpa_bool
6573         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6574     }
6575 }

```

Now, you determine the last empty cell in the row of number 1.

```

6576     \bool_set_false:N \l_tmpa_bool
6577     \int_zero_new:N \l_@@_last_empty_column_int
6578     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6579     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6580     {
6581         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6582         \bool_lazy_or:nnTF
6583         \l_tmpb_bool
6584         {
6585             \cs_if_exist_p:c
6586             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6587         }
6588         { \bool_set_true:N \l_tmpa_bool }
6589         {
6590             \bool_if:NF \l_tmpa_bool
6591             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6592         }
6593     }

```

Now, we loop over the rows.

```

6594     \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int
6595     {

```

We treat the row number ##1 with another loop.

```

6596         \bool_set_false:N \l_tmpa_bool
6597         \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int
6598         {
6599             \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6600             \bool_lazy_or:nnTF
6601             \l_tmpb_bool
6602             {
6603                 \cs_if_exist_p:c
6604                 { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6605             }
6606             { \bool_set_true:N \l_tmpa_bool }
6607             {
6608                 \bool_if:NF \l_tmpa_bool
6609                 {
6610                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6611                     \seq_put_right:Nn
6612                     \l_@@_corners_cells_seq
6613                     { ##1 - #####1 }
6614                 }
6615             }
6616         }
6617     }
6618 }

```

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`).

```

6619 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6620 {
6621     \int_set:Nn \l_tmpa_int { #1 }
6622     \int_set:Nn \l_tmpb_int { #2 }
6623     \bool_set_false:N \l_tmpb_bool
6624     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6625     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6626 }

```

```

6627 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6628 {
6629   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6630   {
6631     \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6632     {
6633       \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6634       {
6635         \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6636         { \bool_set_true:N \l_tmpb_bool }
6637       }
6638     }
6639   }
6640 }

```

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.

```

6641 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6642 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6643 {
6644   auto-columns-width .code:n =
6645   {
6646     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6647     \dim_gzero_new:N \g_@@_max_cell_width_dim
6648     \bool_set_true:N \l_@@_auto_columns_width_bool
6649   }
6650 }

6651 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6652 {
6653   \int_gincr:N \g_@@_NiceMatrixBlock_int
6654   \dim_zero:N \l_@@_columns_width_dim
6655   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6656   \bool_if:NT \l_@@_block_auto_columns_width_bool
6657   {
6658     \cs_if_exist:cT
6659     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6660     {
6661       % is \exp_args:NNe mandatory?
6662       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6663       {
6664         \use:c
6665         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6666       }
6667     }
6668   }
6669 }

```

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).

```

6670 {
6671   \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.

```

6672 { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6673 {
6674   \bool_if:NT \l_@@_block_auto_columns_width_bool
6675   {
6676     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6677     \iow_shipout:Nx \@mainaux
6678     {
6679       \cs_gset:cpn
6680       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

6681       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6682     }
6683     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6684   }
6685 }
6686 \ignorespacesafterend
6687 }

```

26 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

6688 \cs_generate_variant:Nn \dim_min:nn { v n }
6689 \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).

```

6690 \cs_new_protected:Npn \@@_create_extra_nodes:
6691 {
6692   \bool_if:nTF \l_@@_medium_nodes_bool
6693   {
6694     \bool_if:nTF \l_@@_large_nodes_bool
6695     \@@_create_medium_and_large_nodes:
6696     \@@_create_medium_nodes:
6697   }
6698   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6699 }

```

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`.

```

6700 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6701 {
6702   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6703   {
6704     \dim_zero_new:c { \l_@@_row_\@@_i: _min_dim }
6705     \dim_set_eq:cN { \l_@@_row_\@@_i: _min_dim } \c_max_dim
6706     \dim_zero_new:c { \l_@@_row_\@@_i: _max_dim }
6707     \dim_set:cn { \l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6708   }
6709   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6710   {
6711     \dim_zero_new:c { \l_@@_column_\@@_j: _min_dim }
6712     \dim_set_eq:cN { \l_@@_column_\@@_j: _min_dim } \c_max_dim
6713     \dim_zero_new:c { \l_@@_column_\@@_j: _max_dim }
6714     \dim_set:cn { \l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6715   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6716   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6717   {
6718     \int_step_variable:nnNn
6719     \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.

```

6720     {
6721       \cs_if_exist:cT
6722       { \pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor `southwest` of the (normal) node of the cell (i - j). They will be stored in `\pgf@x` and `\pgf@y`.

```

6723     {
6724       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6725       \dim_set:cn { \l_@@_row_\@@_i: _min_dim }
6726       { \dim_min:vn { \l_@@_row _ \@@_i: _min_dim } \pgf@y }
6727       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6728       {
6729         \dim_set:cn { \l_@@_column _ \@@_j: _min_dim }
6730         { \dim_min:vn { \l_@@_column _ \@@_j: _min_dim } \pgf@x }
6731       }

```

We retrieve the coordinates of the anchor `northeast` of the (normal) node of the cell (i - j). They will be stored in `\pgf@x` and `\pgf@y`.

```

6732       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6733       \dim_set:cn { \l_@@_row _ \@@_i: _ max_dim }
6734       { \dim_max:vn { \l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6735       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6736       {
6737         \dim_set:cn { \l_@@_column _ \@@_j: _ max_dim }
6738         { \dim_max:vn { \l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6739       }
6740     }
6741   }
6742 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6743   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6744   {
6745     \dim_compare:nNnT
6746     { \dim_use:c { \l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6747     {
6748       \@@_qpoint:n { row - \@@_i: - base }

```

```

6749         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6750         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6751     }
6752 }
6753 \int_step_variable:nNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6754 {
6755     \dim_compare:nNnT
6756     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6757     {
6758         \@@_qpoint:n { col - \@@_j: }
6759         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6760         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6761     }
6762 }
6763 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6764 \cs_new_protected:Npn \@@_create_medium_nodes:
6765 {
6766     \pgfpicture
6767     \pgfrememberpicturepositiononpagetrue
6768     \pgf@relevantforpicturesizefalse
6769     \@@_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”.

```

6770     \tl_set:Nn \l_@@_suffix_tl { -medium }
6771     \@@_create_nodes:
6772     \endpgfpicture
6773 }

```

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:`.

```

6774 \cs_new_protected:Npn \@@_create_large_nodes:
6775 {
6776     \pgfpicture
6777     \pgfrememberpicturepositiononpagetrue
6778     \pgf@relevantforpicturesizefalse
6779     \@@_computations_for_medium_nodes:
6780     \@@_computations_for_large_nodes:
6781     \tl_set:Nn \l_@@_suffix_tl { - large }
6782     \@@_create_nodes:
6783     \endpgfpicture
6784 }

6785 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6786 {
6787     \pgfpicture
6788     \pgfrememberpicturepositiononpagetrue
6789     \pgf@relevantforpicturesizefalse
6790     \@@_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”.

```

6791     \tl_set:Nn \l_@@_suffix_tl { - medium }
6792     \@@_create_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6793     \@@_computations_for_large_nodes:
6794     \tl_set:Nn \l_@@_suffix_tl { - large }
6795     \@@_create_nodes:
6796     \endpgfpicture
6797 }

```

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.

```

6798 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6799 {
6800     \int_set:Nn \l_@@_first_row_int 1
6801     \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`.

```

6802     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6803     {
6804         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6805         {
6806             (
6807                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6808                 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6809             )
6810             / 2
6811         }
6812         \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6813         { l_@@_row _ \@@_i: _ min_dim }
6814     }
6815     \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6816     {
6817         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6818         {
6819             (
6820                 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6821                 \dim_use:c
6822                 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6823             )
6824             / 2
6825         }
6826         \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6827         { l_@@_column _ \@@_j: _ max _ dim }
6828     }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6829     \dim_sub:cn
6830     { l_@@_column _ 1 _ min _ dim }
6831     \l_@@_left_margin_dim
6832     \dim_add:cn
6833     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6834     \l_@@_right_margin_dim
6835 }

```

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).

```

6836 \cs_new_protected:Npn \@@_create_nodes:
6837 {
6838     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6839     {

```



```

6840 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6841 {

```

We draw the rectangular node for the cell ($\@@_i$ - $\@@_j$).

```

6842 \@@_pgf_rect_node:nnnnn
6843 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6844 { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6845 { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6846 { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6847 { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6848 \str_if_empty:NF \l_@@_name_str
6849 {
6850 \pgfnodealias
6851 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6852 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6853 }
6854 }
6855 }

```

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 .

```

6856 \seq_map_pairwise_function:NNN
6857 \g_@@_multicolumn_cells_seq
6858 \g_@@_multicolumn_sizes_seq
6859 \@@_node_for_multicolumn:nn
6860 }

```

```

6861 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6862 {
6863 \cs_set_nopar:Npn \@@_i: { #1 }
6864 \cs_set_nopar:Npn \@@_j: { #2 }
6865 }

```

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”).

```

6866 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6867 {
6868 \@@_extract_coords_values: #1 \q_stop
6869 \@@_pgf_rect_node:nnnnn
6870 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6871 { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6872 { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6873 { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
6874 { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6875 \str_if_empty:NF \l_@@_name_str
6876 {
6877 \pgfnodealias
6878 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6879 { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6880 }
6881 }

```

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.

```

6882 \keys_define:nn { NiceMatrix / Block / FirstPass }
6883 {
6884   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6885   l .value_forbidden:n = true ,
6886   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6887   r .value_forbidden:n = true ,
6888   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6889   c .value_forbidden:n = true ,
6890   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6891   L .value_forbidden:n = true ,
6892   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6893   R .value_forbidden:n = true ,
6894   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6895   C .value_forbidden:n = true ,
6896   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6897   t .value_forbidden:n = true ,
6898   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6899   T .value_forbidden:n = true ,
6900   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6901   b .value_forbidden:n = true ,
6902   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6903   B .value_forbidden:n = true ,
6904   color .code:n =
6905     \@@_color:n { #1 }
6906     \tl_set_rescan:Nnn
6907       \l_@@_draw_tl
6908       { \char_set_catcode_other:N ! }
6909       { #1 } ,
6910   color .value_required:n = true ,
6911   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6912   respect-arraystretch .default:n = true
6913 }

```

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.

```

6914 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

6915 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6916 {

```

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).

```

6917   \peek_remove_spaces:n
6918   {
6919     \tl_if_blank:nTF { #2 }
6920     { \@@_Block_i 1-1 \q_stop }
6921     {
6922       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6923       \@@_Block_i_czech \@@_Block_i
6924       #2 \q_stop
6925     }
6926     { #1 } { #3 } { #4 }
6927   }
6928 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6929 \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`.

```

6930 {
6931   \char_set_catcode_active:N -
6932   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6933 }

```

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.

```

6934 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6935 {

```

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).

```

6936   \bool_lazy_or:nnTF
6937     { \tl_if_blank_p:n { #1 } }
6938     { \str_if_eq_p:nn { #1 } { * } }
6939     { \int_set:Nn \l_tmpa_int { 100 } }
6940     { \int_set:Nn \l_tmpa_int { #1 } }
6941   \bool_lazy_or:nnTF
6942     { \tl_if_blank_p:n { #2 } }
6943     { \str_if_eq_p:nn { #2 } { * } }
6944     { \int_set:Nn \l_tmpb_int { 100 } }
6945     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6946   \int_compare:nNnTF \l_tmpb_int = 1
6947   {
6948     \str_if_empty:NTF \l_@@_hpos_cell_str
6949     { \str_set:Nn \l_@@_hpos_block_str c }
6950     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6951   }
6952   { \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.

```

6953   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6954   \tl_set:Nx \l_tmpa_tl
6955   {
6956     { \int_use:N \c@iRow }
6957     { \int_use:N \c@jCol }
6958     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6959     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6960   }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

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

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by currying).

```

6961   \bool_if:nTF
6962   {
6963     (
6964       \int_compare_p:nNn { \l_tmpa_int } = 1
6965       ||

```

```

6966         \int_compare_p:nNn { \l_tmpb_int } = 1
6967     )
6968     && ! \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.

6969     && ! \l_@@_X_bool
6970 }
6971 { \exp_args:Nee \@@_Block_iv:nnnnn }
6972 { \exp_args:Nee \@@_Block_v:nnnnn }
6973 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6974 }

```

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.

```

6975 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6976 {
6977     \int_gincr:N \g_@@_block_box_int
6978     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6979     {
6980         \tl_gput_right:Nx \g_@@_pre_code_after_tl
6981         {
6982             \@@_actually_diagbox:nnnnnn
6983             { \int_use:N \c@iRow }
6984             { \int_use:N \c@jCol }
6985             { \int_eval:n { \c@iRow + #1 - 1 } }
6986             { \int_eval:n { \c@jCol + #2 - 1 } }
6987             { \g_@@_row_style_tl \exp_not:n { ##1 } }
6988             { \g_@@_row_style_tl \exp_not:n { ##2 } }
6989         }
6990     }
6991     \box_gclear_new:c
6992     { \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!

```

6993     \hbox_gset:cn
6994     { \g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
6995     {

```

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}`).

```

6996     \tl_if_empty:NTF \l_@@_color_tl
6997     { \int_compare:nNnT { #2 } = 1 \set@color }
6998     { \@@_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`.

```

6999 \int_compare:nNnT { #1 } = 1
7000 {
7001   \int_if_zero:nTF \c@iRow
7002     \l_@@_code_for_first_row_tl
7003     {
7004       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7005       \l_@@_code_for_last_row_tl
7006     }
7007   \g_@@_row_style_tl
7008 }
7009 \bool_if:NF \l_@@_respect_arraystretch_bool
7010 { \cs_set:Npn \arraystretch { 1 } }
7011 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```
7012 #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.).

```
7013 \@@_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}`.

```

7014 \bool_if:NTF \l_@@_tabular_bool
7015 {
7016   \bool_lazy_all:nTF
7017     {
7018       { \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.

```

7019     { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
7020     { ! \g_@@_rotate_bool }
7021   }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7022 {
7023   \use:e
7024   {
7025     \exp_not:N \begin { minipage }%
7026     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7027     { \l_@@_col_width_dim }
7028     \str_case:Vn \l_@@_hpos_block_str
7029       { c \centering r \raggedleft l \raggedright }
7030   }
7031   #5
7032   \end { minipage }
7033 }

```

In the other cases, we use a `{tabular}`.

```

7034 {
7035   \use:e
7036   {
7037     \exp_not:N \begin { tabular }%
7038     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7039     { @ { } \l_@@_hpos_block_str @ { } }
7040   }
7041   #5
7042   \end { tabular }
7043 }
7044 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7045     {
7046         \c_math_toggle_token
7047         \use:e
7048         {
7049             \exp_not:N \begin { array }%
7050             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
7051             { @ { } \l_@@_hpos_block_str @ { } }
7052         }
7053         #5
7054         \end { array }
7055         \c_math_toggle_token
7056     }
7057 }

```

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).

```

7058 \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.

```

7059 \int_compare:nNnT { #2 } = 1
7060 {
7061     \dim_gset:Nn \g_@@_blocks_wd_dim
7062     {
7063         \dim_max:nn
7064         \g_@@_blocks_wd_dim
7065         {
7066             \box_wd:c
7067             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7068         }
7069     }
7070 }

```

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.

```

7071 \str_if_eq:VnT \l_@@_vpos_of_block_str { c }
7072 {
7073     \int_compare:nNnT { #1 } = 1
7074     {
7075         \dim_gset:Nn \g_@@_blocks_ht_dim
7076         {
7077             \dim_max:nn
7078             \g_@@_blocks_ht_dim
7079             {
7080                 \box_ht:c
7081                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7082             }
7083         }
7084         \dim_gset:Nn \g_@@_blocks_dp_dim
7085         {
7086             \dim_max:nn
7087             \g_@@_blocks_dp_dim
7088             {
7089                 \box_dp:c
7090                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

¹⁵If the block has a key of a vertical position, that means that it has to be put in a vertical space determined by the *others* cells of the row. Therefore there is no point creating space here. Moreover, that would lead to problems when a multi-row block with a position key such as `b` or `B`.

```

7091     }
7092   }
7093 }
7094 }
7095 \seq_gput_right:Nx \g_@@_blocks_seq
7096 {
7097   \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.

```

7098   {
7099     \exp_not:n { #3 } ,
7100     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7101     \bool_if:NT \g_@@_rotate_bool
7102     {
7103       \bool_if:NTF \g_@@_rotate_c_bool
7104       { v-center }
7105       { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7106     }
7107   }
7108   {
7109     \box_use_drop:c
7110     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7111   }
7112 }
7113 }
7114 \bool_set_false:N \g_@@_rotate_c_bool
7115 }

```

```

7116 \cs_new:Npn \@@_adjust_hpos_rotate:
7117 {
7118   \bool_if:NT \g_@@_rotate_bool
7119   {
7120     \str_set:Nx \l_@@_hpos_block_str
7121     {
7122       \bool_if:NTF \g_@@_rotate_c_bool
7123       { c }
7124       {
7125         \str_case:VnF \l_@@_vpos_of_block_str
7126         { b l B l t r T r }
7127         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7128       }
7129     }
7130   }
7131 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7132 \cs_new_protected:Npn \@@_rotate_box_of_block:
7133 {
7134   \box_grotate:cn
7135   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7136   { 90 }
7137   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7138   {
7139     \vbox_gset_top:cn
7140     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7141     {

```

```

7142         \skip_vertical:n { 0.8 ex }
7143         \box_use:c
7144         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7145     }
7146 }
7147 \bool_if:NT \g_@@_rotate_c_bool
7148 {
7149     \hbox_gset:cn
7150     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7151     {
7152         \c_math_toggle_token
7153         \vcenter
7154         {
7155             \box_use:c
7156             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7157         }
7158         \c_math_toggle_token
7159     }
7160 }
7161 }

```

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.

```

7162 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7163 {
7164     \seq_gput_right:Nx \g_@@_blocks_seq
7165     {
7166         \l_tmpa_tl
7167         { \exp_not:n { #3 } }
7168         {
7169             \bool_if:NTF \l_@@_tabular_bool
7170             {
7171                 \group_begin:
7172                 \bool_if:NF \l_@@_respect_arraystretch_bool
7173                 { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7174                 \exp_not:n
7175                 {
7176                     \dim_zero:N \extrarowheight
7177                     #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).

```

7178         % \@@_adjust_hpos_rotate:
7179         \use:e
7180         {
7181             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
7182             { @ { } \l_@@_hpos_block_str @ { } }
7183         }
7184         #5
7185         \end { tabular }
7186     }
7187     \group_end:
7188 }

```


When we are *not* in an environments {NiceTabular} (or similar).

```

7189         {
7190             \group_begin:
7191             \bool_if:NF \l_@@_respect_arraystretch_bool
7192             { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7193             \exp_not:n
7194             {
7195                 \dim_zero:N \extrarowheight
7196                 #4
7197                 % \@@_adjust_hpos_rotate:
7198                 \c_math_toggle_token % :n c
7199                 \use:e
7200                 {
7201                     \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
7202                     { @ { } \l_@@_hpos_block_str @ { } }
7203                 }
7204                 #5
7205                 \end { array }
7206                 \c_math_toggle_token
7207             }
7208             \group_end:
7209         }
7210     }
7211 }
7212 }
```

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).

```

7213 \keys_define:nn { NiceMatrix / Block / SecondPass }
7214 {
7215     tikz .code:n =
7216         \IfPackageLoadedTF { tikz }
7217         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7218         { \@@_error:n { tikz-key-without~tikz } } ,
7219     tikz .value_required:n = true ,
7220     fill .code:n =
7221         \tl_set_rescan:Nnn
7222         \l_@@_fill_tl
7223         { \char_set_catcode_other:N ! }
7224         { #1 } ,
7225     fill .value_required:n = true ,
7226     opacity .tl_set:N = \l_@@_opacity_tl ,
7227     opacity .value_required:n = true ,
7228     draw .code:n =
7229         \tl_set_rescan:Nnn
7230         \l_@@_draw_tl
7231         { \char_set_catcode_other:N ! }
7232         { #1 } ,
7233     draw .default:n = default ,
7234     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7235     rounded-corners .default:n = 4 pt ,
7236     color .code:n =
7237         \@@_color:n { #1 }
7238         \tl_set_rescan:Nnn
7239         \l_@@_draw_tl
7240         { \char_set_catcode_other:N ! }
7241         { #1 } ,
7242     borders .clist_set:N = \l_@@_borders_clist ,
7243     borders .value_required:n = true ,
7244     hvlines .meta:n = { vlines , hlines } ,
7245     vlines .bool_set:N = \l_@@_vlines_block_bool,
```

```

7246     vlines .default:n = true ,
7247     hlines .bool_set:N = \l_@@_hlines_block_bool,
7248     hlines .default:n = true ,
7249     line-width .dim_set:N = \l_@@_line_width_dim ,
7250     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7251     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7252     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7253     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7254     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7255         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7256     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7257         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7258     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7259         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7260     t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
7261     T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
7262     b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
7263     B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
7264     v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
7265     v-center .value_forbidden:n = true ,
7266     name .tl_set:N = \l_@@_block_name_str ,
7267     name .value_required:n = true ,
7268     name .initial:n = ,
7269     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
7270     transparent .bool_set:N = \l_@@_transparent_bool ,
7271     transparent .default:n = true ,
7272     transparent .initial:n = false ,
7273     unknown .code:n = \@_error:n { Unknown~key~for~Block }
7274 }

```

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.

```

7275 \cs_new_protected:Npn \@@_draw_blocks:
7276 {
7277     \cs_set_eq:NN \ialign \@@_old_ialign:
7278     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7279 }
7280 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7281 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7282     \int_zero_new:N \l_@@_last_row_int
7283     \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.

```

7284     \int_compare:nNnTF { #3 } > { 99 }
7285     { \int_set_eq:NN \l_@@_last_row_int \c{iRow} }
7286     { \int_set:Nn \l_@@_last_row_int { #3 } }
7287     \int_compare:nNnTF { #4 } > { 99 }
7288     { \int_set_eq:NN \l_@@_last_col_int \c{jCol} }
7289     { \int_set:Nn \l_@@_last_col_int { #4 } }
7290     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7291     {
7292         \bool_lazy_and:nnTF

```

```

7293 \l_@@_preamble_bool
7294 {
7295   \int_compare_p:n
7296   { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7297 }
7298 {
7299   \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
7300   \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
7301   \@@_msg_redirect_name:nn { columns~not~used } { none }
7302 }
7303 { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7304 }
7305 {
7306   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7307   { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7308   { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7309 }
7310 }

```

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

```

7311 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7312 {

```

The group is for the keys.

```

7313   \group_begin:
7314   \int_compare:nNnT { #1 } = { #3 }
7315   { \str_set:Nn \l_@@_vpos_of_block_str { t } }
7316   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7317   \bool_if:NT \l_@@_vlines_block_bool
7318   {
7319     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7320     {
7321       \@@_vlines_block:nnn
7322       { \exp_not:n { #5 } }
7323       { #1 - #2 }
7324       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7325     }
7326   }
7327   \bool_if:NT \l_@@_hlines_block_bool
7328   {
7329     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7330     {
7331       \@@_hlines_block:nnn
7332       { \exp_not:n { #5 } }
7333       { #1 - #2 }
7334       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7335     }
7336   }
7337   \bool_if:nF
7338   {
7339     \l_@@_transparent_bool
7340     || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7341   }
7342   {

```

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).

```

7343   \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7344   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7345 }

```

```

7346 \bool_lazy_and:nnT
7347 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
7348 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7349 { \@@_error:n { hlines-with-color } } }

7350 \tl_if_empty:NF \l_@@_draw_tl
7351 {
7352   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7353   {
7354     \@@_stroke_block:nnn
7355     { \exp_not:n { #5 } } % #5 are the options
7356     { #1 - #2 }
7357     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7358   }
7359   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7360   { { #1 } { #2 } { #3 } { #4 } }
7361 }

7362 \clist_if_empty:NF \l_@@_borders_clist
7363 {
7364   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7365   {
7366     \@@_stroke_borders_block:nnn
7367     { \exp_not:n { #5 } }
7368     { #1 - #2 }
7369     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7370   }
7371 }

7372 \tl_if_empty:NF \l_@@_fill_tl
7373 {
7374   \tl_if_empty:NF \l_@@_opacity_tl
7375   {
7376     \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7377       {
7378         \tl_set:Nx \l_@@_fill_tl
7379         {
7380           [ opacity = \l_@@_opacity_tl ,
7381           \tl_tail:V \l_@@_fill_tl
7382         }
7383       }
7384       {
7385         \tl_set:Nx \l_@@_fill_tl
7386         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7387       }
7388     }
7389     \tl_gput_right:Nx \g_@@_pre_code_before_tl
7390     {
7391       \exp_not:N \roundedrectanglecolor
7392       \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7393         { \l_@@_fill_tl }
7394         { { \l_@@_fill_tl } }
7395         { #1 - #2 }
7396         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7397         { \dim_use:N \l_@@_rounded_corners_dim }
7398       }
7399     }
7400   }
7401   \seq_if_empty:NF \l_@@_tikz_seq
7402   {
7403     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7404     {
7405       \@@_block_tikz:nnnnn
7406       { #1 }
7407       { #2 }

```

```

7407         { \int_use:N \l_@@_last_row_int }
7408         { \int_use:N \l_@@_last_col_int }
7409         { \seq_use:Nn \l_@@_tikz_seq { , } }
7410     }
7411 }

7412 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7413 {
7414     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7415     {
7416         \@@_actually_diagbox:nnnnnn
7417         { #1 }
7418         { #2 }
7419         { \int_use:N \l_@@_last_row_int }
7420         { \int_use:N \l_@@_last_col_int }
7421         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7422     }
7423 }

7424 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7425 \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.

```

7426 \pgfpicture
7427 \pgfrememberpicturepositiononpagetrue
7428 \pgf@relevantforpicturesizefalse
7429 \@@_qpoint:n { row - #1 }
7430 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7431 \@@_qpoint:n { col - #2 }
7432 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7433 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7434 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7435 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7436 \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.

```

7437 \@@_pgf_rect_node:nnnnn
7438 { \@@_env: - #1 - #2 - block }
7439 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7440 \str_if_empty:NF \l_@@_block_name_str
7441 {

```

```

7442 \pgfnodealias
7443 { \l_@@_env: - \l_@@_block_name_str }
7444 { \l_@@_env: - #1 - #2 - block }
7445 \str_if_empty:NF \l_@@_name_str
7446 {
7447 \pgfnodealias
7448 { \l_@@_name_str - \l_@@_block_name_str }
7449 { \l_@@_env: - #1 - #2 - block }
7450 }
7451 }

```

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.

```

7452 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7453 {
7454 \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.

```

7455 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7456 {

```

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.

```

7457 \cs_if_exist:cT
7458 { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
7459 {
7460 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7461 {
7462 \pgfpointanchor { \l_@@_env: - ##1 - #2 } { west }
7463 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7464 }
7465 }
7466 }

```

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.

```

7467 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7468 {
7469 \l_@@_qpoint:n { col - #2 }
7470 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7471 }
7472 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7473 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7474 {
7475 \cs_if_exist:cT
7476 { pgf @ sh @ ns @ \l_@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7477 {
7478 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7479 {
7480 \pgfpointanchor
7481 { \l_@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7482 { east }
7483 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7484 }
7485 }
7486 }
7487 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7488 {
7489 \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7490 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7491 }

```

```

7492 \@@_pgf_rect_node:nnnnn
7493 { \@@_env: - #1 - #2 - block - short }
7494 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7495 }

```

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.

```

7496 \bool_if:NT \l_@@_medium_nodes_bool
7497 {
7498   \@@_pgf_rect_node:nnn
7499   { \@@_env: - #1 - #2 - block - medium }
7500   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7501   {
7502     \pgfpointanchor
7503     { \@@_env:
7504       - \int_use:N \l_@@_last_row_int
7505       - \int_use:N \l_@@_last_col_int - medium
7506     }
7507     { south-east }
7508   }
7509 }

```

Now, we will put the label of the block.

```

7510 \bool_lazy_any:nTF
7511 {
7512   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7513   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7514   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7515 }
7516 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7517 \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 `l`.

```

7518 \bool_if:nT \g_@@_last_col_found_bool
7519 {
7520   \int_compare:nNnT { #2 } = \g_@@_col_total_int
7521   { \str_set:Nn \l_@@_hpos_block_str l }
7522 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7523 \tl_set:Nx \l_tmpa_tl
7524 {
7525   \str_case:Vn \l_@@_vpos_of_block_str
7526   {
7527     c {
7528       \str_case:Vn \l_@@_hpos_block_str
7529       {
7530         c { center }
7531         l { west }
7532         r { east }
7533       }
7534     }
7535   }
7536   T {
7537     \str_case:Vn \l_@@_hpos_block_str
7538     {
7539       c { north }
7540       l { north-west }
7541       r { north-east }
7542     }
7543   }

```

```

7544         }
7545         B {
7546             \str_case:Vn \l_@@_hpos_block_str
7547             {
7548                 c { south}
7549                 l { south~west }
7550                 r { south~east }
7551             }
7552         }
7553     }
7554 }
7555
7556 \pgftransformshift
7557 {
7558     \pgfpointanchor
7559     {
7560         \@@_env: - #1 - #2 - block
7561         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7562     }
7563     { \l_tmpa_tl }
7564 }
7565 \pgfset
7566 {
7567     inner~xsep = \c_zero_dim ,
7568     inner~ysep = \l_@@_block_ysep_dim
7569 }
7570 \pgfnode
7571 { rectangle }
7572 { \l_tmpa_tl }
7573 { \box_use_drop:N \l_@@_cell_box } { } { }
7574 }

```

End of the case when `\l_@@_vpos_of_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

7575 {
7576     \pgfextracty \l_tmpa_dim
7577     {
7578         \@@_qpoint:n
7579         {
7580             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7581             - base
7582         }
7583     }
7584     \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.

```

7585 \pgfpointanchor
7586 {
7587     \@@_env: - #1 - #2 - block
7588     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7589 }
7590 {
7591     \str_case:Vn \l_@@_hpos_block_str
7592     {
7593         c { center }
7594         l { west }
7595         r { east }
7596     }
7597 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7598 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7599 \pgfset { inner~sep = \c_zero_dim }
7600 \pgfnode
7601 { rectangle }

```



```

7602     {
7603         \str_case:Vn \l_@@_hpos_block_str
7604         {
7605             c { base }
7606             l { base~west }
7607             r { base~east }
7608         }
7609     }
7610     { \box_use_drop:N \l_@@_cell_box } { } { }
7611 }
7612 \endpgfpicture
7613 \group_end:
7614 }

```

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).

```

7615 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7616 {
7617     \group_begin:
7618     \tl_clear:N \l_@@_draw_tl
7619     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7620     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7621     \pgfpicture
7622     \pgfrememberpicturepositiononpagetrue
7623     \pgf@relevantforpicturesizefalse
7624     \tl_if_empty:NF \l_@@_draw_tl
7625     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7626         \str_if_eq:VnTF \l_@@_draw_tl { default }
7627         { \CT@arc@ }
7628         { \@@_color:V \l_@@_draw_tl }
7629     }
7630     \pgfsetcornersarced
7631     {
7632         \pgfpoint
7633         { \l_@@_rounded_corners_dim }
7634         { \l_@@_rounded_corners_dim }
7635     }
7636     \@@_cut_on_hyphen:w #2 \q_stop
7637     \bool_lazy_and:nnT
7638     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7639     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7640     {
7641         \@@_qpoint:n { row - \l_tmpa_tl }
7642         \dim_set_eq:NN \l_tmpb_dim \pgf@y
7643         \@@_qpoint:n { col - \l_tmpb_tl }
7644         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7645         \@@_cut_on_hyphen:w #3 \q_stop
7646         \int_compare:nNnT \l_tmpa_tl > \c@iRow
7647         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7648         \int_compare:nNnT \l_tmpb_tl > \c@jCol
7649         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7650         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7651         \dim_set_eq:NN \l_tmpa_dim \pgf@y
7652         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7653         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7654         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7655         \pgfpathrectanglecorners
7656         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7657         { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }

```

```

7658     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7659     { \pgfusepathqstroke }
7660     { \pgfusepath { stroke } }
7661   }
7662   \endpgfpicture
7663   \group_end:
7664 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7665 \keys_define:nn { NiceMatrix / BlockStroke }
7666 {
7667   color .tl_set:N = \l_@@_draw_tl ,
7668   draw .code:n =
7669     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7670   draw .default:n = default ,
7671   line-width .dim_set:N = \l_@@_line_width_dim ,
7672   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7673   rounded-corners .default:n = 4 pt
7674 }

```

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).

```

7675 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7676 {
7677   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7678   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7679   \@@_cut_on_hyphen:w #2 \q_stop
7680   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7681   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7682   \@@_cut_on_hyphen:w #3 \q_stop
7683   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7684   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7685   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7686   {
7687     \use:e
7688     {
7689       \@@_vline:n
7690       {
7691         position = ##1 ,
7692         start = \l_@@_tmpc_tl ,
7693         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7694         total-width = \dim_use:N \l_@@_line_width_dim
7695       }
7696     }
7697   }
7698 }
7699 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7700 {
7701   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7702   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7703   \@@_cut_on_hyphen:w #2 \q_stop
7704   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7705   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7706   \@@_cut_on_hyphen:w #3 \q_stop
7707   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7708   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7709   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7710   {
7711     \use:e
7712     {
7713       \@@_hline:n
7714       {

```

```

7715         position = ##1 ,
7716         start = \l_@@_tmpd_tl ,
7717         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7718         total-width = \dim_use:N \l_@@_line_width_dim
7719     }
7720 }
7721 }
7722 }

```

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).

```

7723 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7724 {
7725     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7726     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7727     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7728     { \@@_error:n { borders~forbidden } }
7729     {
7730         \tl_clear_new:N \l_@@_borders_tikz_tl
7731         \keys_set:nV
7732             { NiceMatrix / OnlyForTikzInBorders }
7733         \l_@@_borders_clist
7734         \@@_cut_on_hyphen:w #2 \q_stop
7735         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7736         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7737         \@@_cut_on_hyphen:w #3 \q_stop
7738         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7739         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7740         \@@_stroke_borders_block_i:
7741     }
7742 }
7743 \hook_gput_code:nnn { begindocument } { . }
7744 {
7745     \cs_new_protected:Npx \@@_stroke_borders_block_i:
7746     {
7747         \c_@@_pgfortikzpicture_tl
7748         \@@_stroke_borders_block_ii:
7749         \c_@@_endpgfortikzpicture_tl
7750     }
7751 }
7752 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7753 {
7754     \pgfrememberpicturepositiononpagetrue
7755     \pgf@relevantforpicturesizefalse
7756     \CT@arc@
7757     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7758     \clist_if_in:NnT \l_@@_borders_clist { right }
7759     { \@@_stroke_vertical:n \l_tmpb_tl }
7760     \clist_if_in:NnT \l_@@_borders_clist { left }
7761     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7762     \clist_if_in:NnT \l_@@_borders_clist { bottom }
7763     { \@@_stroke_horizontal:n \l_tmpa_tl }
7764     \clist_if_in:NnT \l_@@_borders_clist { top }
7765     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7766 }
7767 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7768 {
7769     tikz .code:n =
7770     \cs_if_exist:NTF \tikzpicture
7771     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7772     { \@@_error:n { tikz-in~borders~without~tikz } } ,

```

```

7773   tikz .value_required:n = true ,
7774   top .code:n = ,
7775   bottom .code:n = ,
7776   left .code:n = ,
7777   right .code:n = ,
7778   unknown .code:n = \@@_error:n { bad-border }
7779 }

```

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).

```

7780 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7781 {
7782   \@@_qpoint:n \l_@@_tmpc_tl
7783   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7784   \@@_qpoint:n \l_tmpa_tl
7785   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7786   \@@_qpoint:n { #1 }
7787   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7788   {
7789     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7790     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7791     \pgfusepathqstroke
7792   }
7793   {
7794     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7795     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7796   }
7797 }

```

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).

```

7798 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7799 {
7800   \@@_qpoint:n \l_@@_tmpd_tl
7801   \clist_if_in:NnTF \l_@@_borders_clist { left }
7802   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7803   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7804   \@@_qpoint:n \l_tmpb_tl
7805   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7806   \@@_qpoint:n { #1 }
7807   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7808   {
7809     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7810     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7811     \pgfusepathqstroke
7812   }
7813   {
7814     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7815     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7816   }
7817 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

7818 \keys_define:nn { NiceMatrix / BlockBorders }
7819 {
7820   borders .clist_set:N = \l_@@_borders_clist ,
7821   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7822   rounded-corners .default:n = 4 pt ,
7823   line-width .dim_set:N = \l_@@_line_width_dim
7824 }

```

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.

```

7825 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7826 {
7827   \begin { tikzpicture }
7828   \@@_clip_with_rounded_corners:
7829   \clist_map_inline:nn { #5 }
7830   {
7831     \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7832     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7833     (
7834       [
7835         xshift = \dim_use:N \l_@@_offset_dim ,
7836         yshift = - \dim_use:N \l_@@_offset_dim
7837       ]
7838       #1 -| #2
7839     )
7840     rectangle
7841     (
7842       [
7843         xshift = - \dim_use:N \l_@@_offset_dim ,
7844         yshift = \dim_use:N \l_@@_offset_dim
7845       ]
7846       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7847     ) ;
7848   }
7849   \end { tikzpicture }
7850 }
7851 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7852 \keys_define:nn { NiceMatrix / SpecialOffset }
7853 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7854 \cs_set_protected:Npn \@@_renew_matrix:
7855 {
7856   \RenewDocumentEnvironment { pmatrix } { } { }
7857   { \pNiceMatrix }
7858   { \endpNiceMatrix }
7859   \RenewDocumentEnvironment { vmatrix } { } { }
7860   { \vNiceMatrix }
7861   { \endvNiceMatrix }
7862   \RenewDocumentEnvironment { Vmatrix } { } { }
7863   { \VNiceMatrix }
7864   { \endVNiceMatrix }
7865   \RenewDocumentEnvironment { bmatrix } { } { }
7866   { \bNiceMatrix }
7867   { \endbNiceMatrix }
7868   \RenewDocumentEnvironment { Bmatrix } { } { }
7869   { \BNiceMatrix }
7870   { \endBNiceMatrix }
7871 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7872 \keys_define:nn { NiceMatrix / Auto }
7873 {
7874   columns-type .tl_set:N = \l_@@_columns_type_tl ,
7875   columns-type .value_required:n = true ,
7876   l .meta:n = { columns-type = l } ,
7877   r .meta:n = { columns-type = r } ,
7878   c .meta:n = { columns-type = c } ,
7879   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7880   delimiters / color .value_required:n = true ,
7881   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7882   delimiters / max-width .default:n = true ,
7883   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7884   delimiters .value_required:n = true ,
7885   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7886   rounded-corners .default:n = 4 pt
7887 }
7888 \NewDocumentCommand \AutoNiceMatrixWithDelims
7889 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7890 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7891 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7892 {

```

The group is for the protection of the keys.

```

7893   \group_begin:
7894   \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`).

```

7895   % \bool_set_false:N \l_@@_preamble_bool
7896   \use:e
7897   {
7898     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7899     { * { #4 } { \exp_not:N \l_@@_columns_type_tl } }
7900     [ \exp_not:N \l_tmpa_tl ]
7901   }
7902   \int_if_zero:nT \l_@@_first_row_int
7903   {
7904     \int_if_zero:nT \l_@@_first_col_int { & }
7905     \prg_replicate:nn { #4 - 1 } { & }
7906     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7907   }
7908   \prg_replicate:nn { #3 }
7909   {
7910     \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`).

```

7911     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7912     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7913   }
7914   \int_compare:nNnT \l_@@_last_row_int > { -2 }
7915   {
7916     \int_if_zero:nT \l_@@_first_col_int { & }
7917     \prg_replicate:nn { #4 - 1 } { & }
7918     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7919   }
7920   \end { NiceArrayWithDelims }
7921   \group_end:
7922 }

```

```

7923 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7924 {
7925   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7926   {
7927     \bool_gset_true:N \g_@@_delims_bool
7928     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7929     \AutoNiceMatrixWithDelims { #2 } { #3 }
7930   }
7931 }
7932 \@@_define_com:nnn p ( )
7933 \@@_define_com:nnn b [ ]
7934 \@@_define_com:nnn v | |
7935 \@@_define_com:nnn V \| \|
7936 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7937 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7938 {
7939   \group_begin:
7940   \bool_gset_false:N \g_@@_delims_bool
7941   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7942   \group_end:
7943 }

```

30 The redefinition of the command `\dotfill`

```

7944 \cs_set_eq:NN \@@_old_dotfill \dotfill
7945 \cs_new_protected:Npn \@@_dotfill:
7946 {

```

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}`).

```

7947   \@@_old_dotfill
7948   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7949 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

7950 \cs_new_protected:Npn \@@_dotfill_i:
7951 { \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.

```

7952 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7953 {
7954   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7955   {
7956     \@@_actually_diagbox:nnnnnn
7957     { \int_use:N \c_iRow }
7958     { \int_use:N \c_jCol }
7959     { \int_use:N \c_iRow }
7960     { \int_use:N \c_jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

`\@@_if_row_less_than:nn_{number}_{instructions}`

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

7961         { \g_@@_row_style_tl \exp_not:n { #1 } }
7962         { \g_@@_row_style_tl \exp_not:n { #2 } }
7963     }

```

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`.

```

7964     \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7965     {
7966         { \int_use:N \c@iRow }
7967         { \int_use:N \c@jCol }
7968         { \int_use:N \c@iRow }
7969         { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7970         { }
7971     }
7972 }

```

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.

```

7973 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7974 {
7975     \pgfpicture
7976     \pgf@relevantforpicturesizefalse
7977     \pgfrememberpicturepositiononpagetrue
7978     \@@_qpoint:n { row - #1 }
7979     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7980     \@@_qpoint:n { col - #2 }
7981     \dim_set_eq:NN \l_tmpb_dim \pgf@x
7982     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7983     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7984     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7985     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7986     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7987     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7988     {

```

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.

```

7989     \CT@arc@
7990     \pgfsetroundcap
7991     \pgfusepathqstroke
7992 }
7993 \pgfset { inner~sep = 1 pt }
7994 \pgfscope
7995 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7996 \pgfnode { rectangle } { south-west }
7997 {
7998     \begin { minipage } { 20 cm }
7999     \@@_math_toggle_token: #5 \@@_math_toggle_token:
8000     \end { minipage }
8001 }
8002 { }
8003 { }
8004 \endpgfscope
8005 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }

```



```

8006 \pgfnode { rectangle } { north-east }
8007 {
8008   \begin { minipage } { 20 cm }
8009   \raggedleft
8010   \@@_math_toggle_token: #6 \@@_math_toggle_token:
8011   \end { minipage }
8012 }
8013 { }
8014 { }
8015 \endpgfpicture
8016 }

```

32 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 82.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8017 \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 `\`.

```

8018 \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`.

```

8019 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8020 {
8021   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8022   \@@_CodeAfter_iv:n
8023 }

```

We catch the argument of the command `\end` (in `#1`).

```

8024 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8025 {

```

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.

```

8026   \str_if_eq:eeTF \@currentvir { #1 }
8027   { \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`.

```

8028   {
8029     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8030     \@@_CodeAfter_ii:n
8031   }
8032 }

```

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_t1` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

8033 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8034 {
8035   \pgfpicture
8036   \pgfrememberpicturepositiononpagetrue
8037   \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.

```

8038   \@@_qpoint:n { row - 1 }
8039   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8040   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8041   \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).

```

8042   \bool_if:nTF { #3 }
8043   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8044   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8045   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8046   {
8047     \cs_if_exist:cT
8048     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8049     {
8050       \pgfpointanchor
8051       { \@@_env: - ##1 - #2 }
8052       { \bool_if:nTF { #3 } { west } { east } }
8053       \dim_set:Nn \l_tmpa_dim
8054       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8055     }
8056   }

```

Now we can put the delimiter with a node of PGF.

```

8057   \pgfset { inner~sep = \c_zero_dim }
8058   \dim_zero:N \nulldelimiterspace
8059   \pgftransformshift
8060   {
8061     \pgfpoint
8062     { \l_tmpa_dim }
8063     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8064   }
8065   \pgfnode
8066   { rectangle }
8067   { \bool_if:nTF { #3 } { east } { west } }
8068   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8069     \nullfont
8070     \c_math_toggle_token
8071     \@@_color:V \l_@@_delimiters_color_t1
8072     \bool_if:nTF { #3 } { \left #1 } { \left . }

```

```

8073     \vcenter
8074     {
8075         \nullfont
8076         \hrule \@height
8077             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8078             \@depth \c_zero_dim
8079             \@width \c_zero_dim
8080     }
8081     \bool_if:nTF { #3 } { \right . } { \right #1 }
8082     \c_math_toggle_token
8083 }
8084 { }
8085 { }
8086 \endpgfpicture
8087 }

```

34 The command \SubMatrix

```

8088 \keys_define:nn { NiceMatrix / sub-matrix }
8089 {
8090     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8091     extra-height .value_required:n = true ,
8092     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8093     left-xshift .value_required:n = true ,
8094     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8095     right-xshift .value_required:n = true ,
8096     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8097     xshift .value_required:n = true ,
8098     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8099     delimiters / color .value_required:n = true ,
8100     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8101     slim .default:n = true ,
8102     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8103     hlines .default:n = all ,
8104     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8105     vlines .default:n = all ,
8106     hvlines .meta:n = { hlines, vlines } ,
8107     hvlines .value_forbidden:n = true
8108 }
8109 \keys_define:nn { NiceMatrix }
8110 {
8111     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8112     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8113     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8114     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8115 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8116 \keys_define:nn { NiceMatrix / SubMatrix }
8117 {
8118     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8119     delimiters / color .value_required:n = true ,
8120     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8121     hlines .default:n = all ,
8122     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8123     vlines .default:n = all ,
8124     hvlines .meta:n = { hlines, vlines } ,
8125     hvlines .value_forbidden:n = true ,
8126     name .code:n =

```

```

8127 \tl_if_empty:nTF { #1 }
8128 { \@@_error:n { Invalid-name } }
8129 {
8130   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8131   {
8132     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8133     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8134     {
8135       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8136       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8137     }
8138   }
8139   { \@@_error:n { Invalid-name } }
8140 } ,
8141 name .value_required:n = true ,
8142 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8143 rules .value_required:n = true ,
8144 code .tl_set:N = \l_@@_code_tl ,
8145 code .value_required:n = true ,
8146 unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8147 }

8148 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8149 {
8150   \peek_remove_spaces:n
8151   {
8152     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8153     {
8154       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8155       [
8156         delimiters / color = \l_@@_delimiters_color_tl ,
8157         hlines = \l_@@_submatrix_hlines_clist ,
8158         vlines = \l_@@_submatrix_vlines_clist ,
8159         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8160         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8161         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8162         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8163         #5
8164       ]
8165     }
8166     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8167   }
8168 }

8169 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8170 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8171 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8172 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8173 {
8174   \seq_gput_right:Nx \g_@@_submatrix_seq
8175   {
We use \str_if_eq:nnTF because it is fully expandable.
8176   { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8177   { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8178   { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8179   { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8180 }
8181 }

```

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`.

```

8182 \hook_gput_code:nnn { begindocument } { . }
8183 {
8184   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8185   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8186   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8187     {
8188       \peek_remove_spaces:n
8189       {
8190         \@@_sub_matrix:nnnnnnn
8191         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8192       }
8193     }
8194 }

```

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).

```

8195 \NewDocumentCommand \@@_compute_i_j:nn
8196 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8197 { \@@_compute_i_j:nnnn #1 #2 }
8198 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8199 {
8200   \tl_set:Nn \l_@@_first_i_tl { #1 }
8201   \tl_set:Nn \l_@@_first_j_tl { #2 }
8202   \tl_set:Nn \l_@@_last_i_tl { #3 }
8203   \tl_set:Nn \l_@@_last_j_tl { #4 }
8204   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8205     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8206   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8207     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8208   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8209     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8210   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8211     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8212 }
8213 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8214 {
8215   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8216 \@@_compute_i_j:nn { #2 } { #3 }
8217 % added 6.19b
8218 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8219 { \cs_set:Npn \arraystretch { 1 } }
8220 \bool_lazy_or:nnTF
8221 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8222 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8223 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8224 {
8225   \str_clear_new:N \l_@@_submatrix_name_str

```

```

8226 \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8227 \pgfpicture
8228 \pgfrememberpicturepositiononpagetrue
8229 \pgf@relevantforpicturesizefalse
8230 \pgfset { inner-sep = \c_zero_dim }
8231 \dim_set_eq:Nn \l_@@_x_initial_dim \c_max_dim
8232 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

8233 \bool_if:NTF \l_@@_submatrix_slim_bool
8234 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8235 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8236 {
8237   \cs_if_exist:cT
8238   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8239   {
8240     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8241     \dim_set:Nn \l_@@_x_initial_dim
8242     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8243   }
8244   \cs_if_exist:cT
8245   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8246   {
8247     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8248     \dim_set:Nn \l_@@_x_final_dim
8249     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8250   }
8251 }
8252 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8253 { \@@_error:nn { Impossible-delimiter } { left } }
8254 {
8255   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8256   { \@@_error:nn { Impossible-delimiter } { right } }
8257   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8258 }
8259 \endpgfpicture
8260 }
8261 \group_end:
8262 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8263 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8264 {
8265   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8266   \dim_set:Nn \l_@@_y_initial_dim
8267   {
8268     \fp_to_dim:n
8269     {
8270       \pgf@y
8271       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8272     }
8273   } % modified 6.13c
8274   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8275   \dim_set:Nn \l_@@_y_final_dim
8276   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8277   % modified 6.13c
8278   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8279   {
8280     \cs_if_exist:cT
8281     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8282     {
8283       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8284       \dim_set:Nn \l_@@_y_initial_dim

```

```

8285         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8286     }
8287     \cs_if_exist:cT
8288     { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8289     {
8290         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8291         \dim_set:Nn \l_@@_y_final_dim
8292         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8293     }
8294 }
8295 \dim_set:Nn \l_tmpa_dim
8296 {
8297     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8298     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8299 }
8300 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8301     \group_begin:
8302     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8303     \@@_set_CT@arc@:V \l_@@_rules_color_tl
8304     \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`.

```

8305     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8306     {
8307         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8308         {
8309             \int_compare:nNnT
8310             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8311             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8312                 \@@_qpoint:n { col - ##1 }
8313                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8314                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8315                 \pgfusepathqstroke
8316             }
8317         }
8318     }

```

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.

```

8319     \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
8320     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8321     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8322     {
8323         \bool_lazy_and:nnTF
8324         { \int_compare_p:nNn { ##1 } > 0 }
8325         {
8326             \int_compare_p:nNn
8327             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8328         {
8329             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8330             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8331             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8332             \pgfusepathqstroke
8333         }
8334         { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
8335     }

```

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.

```

8336 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
8337 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8338 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8339 {
8340   \bool_lazy_and:nnTF
8341   { \int_compare_p:nNn { ##1 } > 0 }
8342   {
8343     \int_compare_p:nNn
8344     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8345   {
8346     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8347   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8348   \dim_set:Nn \l_tmpa_dim
8349   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8350   \str_case:nn { #1 }
8351   {
8352     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8353     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8354     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8355     }
8356   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

8357   \dim_set:Nn \l_tmpb_dim
8358   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8359   \str_case:nn { #2 }
8360   {
8361     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8362     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8363     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8364   }
8365   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8366   \pgfusepathqstroke
8367   \group_end:
8368 }
8369 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8370 }

```

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).

```

8371 \str_if_empty:NF \l_@@_submatrix_name_str
8372 {
8373   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8374   \l_@@_x_initial_dim \l_@@_y_initial_dim
8375   \l_@@_x_final_dim \l_@@_y_final_dim
8376 }
8377 \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`.

```

8378 \begin { pgfscope }
8379 \pgftransformshift
8380 {
8381   \pgfpoint
8382   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8383   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8384 }

```



```

8385 \str_if_empty:NTF \l_@@_submatrix_name_str
8386 { \@@_node_left:nn #1 { } }
8387 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8388 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8389 \pgftransformshift
8390 {
8391   \pgfpoint
8392   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8393   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8394 }
8395 \str_if_empty:NTF \l_@@_submatrix_name_str
8396 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8397 {
8398   \@@_node_right:nnnn #2
8399   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8400 }
8401 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8402 \flag_clear_new:n { nicematrix }
8403 \l_@@_code_tl
8404 }

```

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`.

```

8405 \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.

```

8406 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8407 {
8408   \use:e
8409   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8410 }

```

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 -.

```

8411 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8412 { #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.

```

8413 \tl_const:Nn \c_@@_integers_alist_tl
8414 {
8415   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8416   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8417   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8418   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8419 }

```

```

8420 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8421 {

```

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`.

```

8422 \tl_if_empty:nTF { #2 }
8423 {
8424   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8425   {
8426     \flag_raise:n { nicematrix }
8427     \int_if_even:nTF { \flag_height:n { nicematrix } }
8428     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8429     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8430   }
8431   { #1 }
8432 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8433 { \c_@@_pgfpointanchor_iii:w { #1 } #2 }
8434 }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\c_@@_pgfpointanchor_i:nn`).

```

8435 \cs_new:Npn \c_@@_pgfpointanchor_iii:w #1 #2 -
8436 {
8437   \str_case:nnF { #1 }
8438   {
8439     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8440     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8441   }

```

Now the case of a node of the form $i-j$.

```

8442 {
8443   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8444   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8445 }
8446 }

```

The command `\c_@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

8447 \cs_new_protected:Npn \c_@@_node_left:nn #1 #2
8448 {
8449   \pgfnode
8450   { rectangle }
8451   { east }
8452   {
8453     \nullfont
8454     \c_math_toggle_token
8455     \c_@@_color:V \l_@@_delimiters_color_tl
8456     \left #1
8457     \vcenter
8458     {
8459       \nullfont
8460       \hrule \@height \l_tmpa_dim
8461       \c_@@_depth \c_zero_dim
8462       \c_@@_width \c_zero_dim
8463     }
8464     \right .
8465     \c_math_toggle_token
8466   }
8467   { #2 }

```

```

8468     { }
8469 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

8470 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8471 {
8472   \pgfnode
8473   { rectangle }
8474   { west }
8475   {
8476     \nullfont
8477     \c_math_toggle_token
8478     \@@_color:V \l_@@_delimiters_color_tl
8479     \left .
8480     \vcenter
8481     {
8482       \nullfont
8483       \hrule \@height \l_tmpa_dim
8484       \@depth \c_zero_dim
8485       \@width \c_zero_dim
8486     }
8487     \right #1
8488     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8489     ^ { \smash { #4 } }
8490     \c_math_toggle_token
8491   }
8492   { #2 }
8493   { }
8494 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8495 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8496 {
8497   \peek_remove_spaces:n
8498   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8499 }
8500 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8501 {
8502   \peek_remove_spaces:n
8503   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8504 }
8505 \keys_define:nn { NiceMatrix / Brace }
8506 {
8507   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8508   left-shorten .default:n = true ,
8509   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8510   shorten .meta:n = { left-shorten , right-shorten } ,
8511   right-shorten .default:n = true ,
8512   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8513   yshift .value_required:n = true ,
8514   yshift .initial:n = \c_zero_dim ,
8515   color .tl_set:N = \l_tmpa_tl ,
8516   color .value_required:n = true ,

```

```

8517     unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
8518 }

```

#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`.

```

8519 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8520 {
8521     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8522     \@@_compute_i_j:nn { #1 } { #2 }
8523     \bool_lazy_or:nnTF
8524     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8525     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8526     {
8527         \str_if_eq:nnTF { #5 } { under }
8528         { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8529         { \@@_error:nn { Construct-too-large } { \OverBrace } }
8530     }
8531     {
8532         \tl_clear:N \l_tmpa_tl
8533         \keys_set:nn { NiceMatrix / Brace } { #4 }
8534         \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8535         \pgfpicture
8536         \pgfrememberpicturepositiononpagetrue
8537         \pgf@relevantforpicturesizefalse
8538         \bool_if:NT \l_@@_brace_left_shorten_bool
8539         {
8540             \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8541             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8542             {
8543                 \cs_if_exist:cT
8544                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8545                 {
8546                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8547                     \dim_set:Nn \l_@@_x_initial_dim
8548                     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8549                 }
8550             }
8551         }
8552         \bool_lazy_or:nnT
8553         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8554         { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8555         {
8556             \@@_qpoint:n { col - \l_@@_first_j_tl }
8557             \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8558         }
8559         \bool_if:NT \l_@@_brace_right_shorten_bool
8560         {
8561             \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8562             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8563             {
8564                 \cs_if_exist:cT
8565                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8566                 {
8567                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8568                     \dim_set:Nn \l_@@_x_final_dim
8569                     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8570                 }
8571             }
8572         }
8573         \bool_lazy_or:nnT
8574         { \bool_not_p:n \l_@@_brace_right_shorten_bool }

```

```

8575     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8576     {
8577         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8578         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8579     }
8580     \pgfset { inner~sep = \c_zero_dim }
8581     \str_if_eq:nnTF { #5 } { under }
8582     { \@@_underbrace_i:n { #3 } }
8583     { \@@_overbrace_i:n { #3 } }
8584     \endpgfpicture
8585 }
8586 \group_end:
8587 }

```

The argument is the text to put above the brace.

```

8588 \cs_new_protected:Npn \@@_overbrace_i:n #1
8589 {
8590     \@@_qpoint:n { row - \l_@@_first_i_tl }
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     { south }
8600     {
8601         \vbox_top:n
8602         {
8603             \group_begin:
8604             \everycr { }
8605             \halign
8606             {
8607                 \hfil ## \hfil \crrc
8608                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8609                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8610                 \c_math_toggle_token
8611                 \overbrace
8612                 {
8613                     \hbox_to_wd:nn
8614                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8615                     { }
8616                 }
8617                 \c_math_toggle_token
8618                 \cr
8619             }
8620             \group_end:
8621         }
8622     }
8623     { }
8624     { }
8625 }

```

The argument is the text to put under the brace.

```

8626 \cs_new_protected:Npn \@@_underbrace_i:n #1
8627 {
8628     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8629     \pgftransformshift
8630     {
8631         \pgfpoint
8632         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8633         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }

```

```

8634     }
8635     \pgfnode
8636     { rectangle }
8637     { north }
8638     {
8639         \group_begin:
8640         \everycr { }
8641         \vbox:n
8642         {
8643             \halign
8644             {
8645                 \hfil ## \hfil \crr
8646                 \c_math_toggle_token
8647                 \underbrace
8648                 {
8649                     \hbox_to_wd:nn
8650                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8651                     { }
8652                 }
8653                 \c_math_toggle_token
8654                 \cr
8655                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8656                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8657             }
8658         }
8659         \group_end:
8660     }
8661     { }
8662     { }
8663 }

```

36 The command TikzEveryCell

```

8664 \bool_new:N \l_@@_not_empty_bool
8665 \bool_new:N \l_@@_empty_bool
8666
8667 \keys_define:nn { NiceMatrix / TikzEveryCell }
8668 {
8669     not-empty .code:n =
8670         \bool_lazy_or:nnTF
8671         \l_@@_in_code_after_bool
8672         \g_@@_recreate_cell_nodes_bool
8673         { \bool_set_true:N \l_@@_not_empty_bool }
8674         { \@@_error:n { detection-of-empty-cells } } ,
8675     not-empty .value_forbidden:n = true ,
8676     empty .code:n =
8677         \bool_lazy_or:nnTF
8678         \l_@@_in_code_after_bool
8679         \g_@@_recreate_cell_nodes_bool
8680         { \bool_set_true:N \l_@@_empty_bool }
8681         { \@@_error:n { detection-of-empty-cells } } ,
8682     empty .value_forbidden:n = true ,
8683     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
8684 }
8685
8686
8687 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8688 {
8689     \IfPackageLoadedTF { tikz }

```

```

8690 {
8691   \group_begin:
8692   \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.

```

8693   \tl_set:Nn \l_tmpa_tl { { #2 } }
8694   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8695     { \@@_for_a_block:nnnnn ##1 }
8696   \@@_all_the_cells:
8697   \group_end:
8698 }
8699 { \@@_error:n { TikzEveryCell~without~tikz } }
8700 }
8701
8702 \tl_new:N \@@_i_tl
8703 \tl_new:N \@@_j_tl
8704
8705 \cs_new_protected:Nn \@@_all_the_cells:
8706 {
8707   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8708   {
8709     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8710     {
8711       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8712       {
8713         \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8714           { \@@_i_tl - \@@_j_tl }
8715         {
8716           \bool_set_false:N \l_tmpa_bool
8717           \cs_if_exist:cTF
8718             { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8719             {
8720               \bool_if:NF \l_@@_empty_bool
8721               { \bool_set_true:N \l_tmpa_bool }
8722             }
8723             {
8724               \bool_if:NF \l_@@_not_empty_bool
8725               { \bool_set_true:N \l_tmpa_bool }
8726             }
8727             \bool_if:NT \l_tmpa_bool
8728             {
8729               \@@_block_tikz:nnnnV
8730               \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8731             }
8732           }
8733         }
8734       }
8735     }
8736   }
8737
8738 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8739 {
8740   \bool_if:NF \l_@@_empty_bool
8741   {
8742     \@@_block_tikz:nnnnV
8743     { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8744   }
8745   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8746 }
8747
8748 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8749 {
8750   \int_step_inline:nnn { #1 } { #3 }

```

```

8751 {
8752   \int_step_inline:nnn { #2 } { #4 }
8753   { \cs_set:cpn { cell - ##1 - #####1 } { } }
8754 }
8755 }

```

37 The command \ShowCellNames

```

8756 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8757 {
8758   \dim_zero_new:N \g_@@_tmpc_dim
8759   \dim_zero_new:N \g_@@_tmpd_dim
8760   \dim_zero_new:N \g_@@_tmpe_dim
8761   \int_step_inline:nn \c@iRow
8762   {
8763     \begin { pgfpicture }
8764     \@@_qpoint:n { row - ##1 }
8765     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8766     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8767     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8768     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8769     \bool_if:NTF \l_@@_in_code_after_bool
8770     \end { pgfpicture }
8771     \int_step_inline:nn \c@jCol
8772     {
8773       \hbox_set:Nn \l_tmpa_box
8774       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8775       \begin { pgfpicture }
8776       \@@_qpoint:n { col - #####1 }
8777       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8778       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8779       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8780       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8781       \endpgfpicture
8782       \end { pgfpicture }
8783       \fp_set:Nn \l_tmpa_fp
8784       {
8785         \fp_min:nn
8786         {
8787           \fp_min:nn
8788           { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8789           { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8790         }
8791         { 1.0 }
8792       }
8793       \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8794       \pgfpicture
8795       \pgfrememberpicturepositiononpagetrue
8796       \pgf@relevantforpicturesizefalse
8797       \pgftransformshift
8798       {
8799         \pgfpoint
8800         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8801         { \dim_use:N \g_tmpa_dim }
8802       }
8803       \pgfnode
8804       { rectangle }
8805       { center }
8806       { \box_use:N \l_tmpa_box }
8807       { }
8808       { }
8809       \endpgfpicture

```



```

8810     }
8811   }
8812 }

8813 \NewDocumentCommand \@@_ShowCellNames { }
8814 {
8815   \bool_if:NT \l_@@_in_code_after_bool
8816   {
8817     \pgfpicture
8818     \pgfrememberpicturepositiononpagetrue
8819     \pgf@relevantforpicturesizefalse
8820     \pgfpathrectanglecorners
8821     { \@@_qpoint:n { 1 } }
8822     {
8823       \@@_qpoint:n
8824       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8825     }
8826     \pgfsetfillopacity { 0.75 }
8827     \pgfsetfillcolor { white }
8828     \pgfusepathqfill
8829     \endpgfpicture
8830   }
8831   \dim_zero_new:N \g_@@_tmpc_dim
8832   \dim_zero_new:N \g_@@_tmpd_dim
8833   \dim_zero_new:N \g_@@_tmpe_dim
8834   \int_step_inline:nn \c@iRow
8835   {
8836     \bool_if:NTF \l_@@_in_code_after_bool
8837     {
8838       \pgfpicture
8839       \pgfrememberpicturepositiononpagetrue
8840       \pgf@relevantforpicturesizefalse
8841     }
8842     { \begin { pgfpicture } }
8843     \@@_qpoint:n { row - ##1 }
8844     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8845     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8846     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8847     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8848     \bool_if:NTF \l_@@_in_code_after_bool
8849     { \endpgfpicture }
8850     { \end { pgfpicture } }
8851     \int_step_inline:nn \c@jCol
8852     {
8853       \hbox_set:Nn \l_tmpa_box
8854       {
8855         \normalfont \Large \sffamily \bfseries
8856         \bool_if:NTF \l_@@_in_code_after_bool
8857         { \color { red } }
8858         { \color { red ! 50 } }
8859         ##1 - ####1
8860       }
8861       \bool_if:NTF \l_@@_in_code_after_bool
8862       {
8863         \pgfpicture
8864         \pgfrememberpicturepositiononpagetrue
8865         \pgf@relevantforpicturesizefalse
8866       }
8867       { \begin { pgfpicture } }
8868       \@@_qpoint:n { col - ####1 }
8869       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8870       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8871       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8872       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

8873 \bool_if:NTF \l_@@_in_code_after_bool
8874 { \endpgfpicture }
8875 { \end { pgfpicture } }
8876 \fp_set:Nn \l_tmpa_fp
8877 {
8878   \fp_min:nn
8879   {
8880     \fp_min:nn
8881     { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8882     { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8883   }
8884   { 1.0 }
8885 }
8886 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8887 \pgfpicture
8888 \pgfrememberpicturepositiononpagetrue
8889 \pgf@relevantforpicturesizefalse
8890 \pgftransformshift
8891 {
8892   \pgfpoint
8893   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8894   { \dim_use:N \g_tmpa_dim }
8895 }
8896 \pgfnode
8897 { rectangle }
8898 { center }
8899 { \box_use:N \l_tmpa_box }
8900 { }
8901 { }
8902 \endpgfpicture
8903 }
8904 }
8905 }

```

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.

```

8906 \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.

```

8907 \bool_new:N \g_@@_footnote_bool
8908 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
8909 {
8910   The-key~'\l_keys_key_str'~is-unknown. \\
8911   That-key~will~be-ignored. \\
8912   For-a~list-of~the~available~keys,~type-H<return>.
8913 }
8914 {
8915   The~available~keys~are~(in~alphabetic~order):~
8916   footnote,~
8917   footnotehyper,~
8918   messages-for-Overleaf,~
8919   no-test-for-array,~
8920   renew-dots,~and~

```

```

8921     renew-matrix.
8922 }
8923 \keys_define:nn { NiceMatrix / Package }
8924 {
8925     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8926     renew-dots .value_forbidden:n = true ,
8927     renew-matrix .code:n = \@@_renew_matrix: ,
8928     renew-matrix .value_forbidden:n = true ,
8929     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
8930     footnote .bool_set:N = \g_@@_footnote_bool ,
8931     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
8932     no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
8933     no-test-for-array .default:n = true ,
8934     unknown .code:n = \@@_error:n { Unknown-key-for-package }
8935 }
8936 \ProcessKeysOptions { NiceMatrix / Package }

8937 \@@_msg_new:nn { footnote-with-footnotehyper-package }
8938 {
8939     You-can't-use-the-option-'footnote'~because-the-package~
8940     footnotehyper-has-already-been-loaded.~
8941     If-you-want,~you-can-use-the-option-'footnotehyper'~and-the-footnotes~
8942     within-the-environments-of~nicematrix-will-be-extracted-with-the-tools~
8943     of-the-package-footnotehyper.\\
8944     The~package~footnote-won't-be-loaded.
8945 }
8946 \@@_msg_new:nn { footnotehyper-with-footnote-package }
8947 {
8948     You-can't-use-the-option-'footnotehyper'~because-the-package~
8949     footnote-has-already-been-loaded.~
8950     If-you-want,~you-can-use-the-option-'footnote'~and-the-footnotes~
8951     within-the-environments-of~nicematrix-will-be-extracted-with-the-tools~
8952     of-the-package-footnote.\\
8953     The~package~footnotehyper-won't-be-loaded.
8954 }

8955 \bool_if:NT \g_@@_footnote_bool
8956 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8957     \IfClassLoadedTF { beamer }
8958     { \bool_set_false:N \g_@@_footnote_bool }
8959     {
8960         \IfPackageLoadedTF { footnotehyper }
8961         { \@@_error:n { footnote-with-footnotehyper-package } }
8962         { \usepackage { footnote } }
8963     }
8964 }
8965 \bool_if:NT \g_@@_footnotehyper_bool
8966 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8967     \IfClassLoadedTF { beamer }
8968     { \bool_set_false:N \g_@@_footnote_bool }
8969     {
8970         \IfPackageLoadedTF { footnote }
8971         { \@@_error:n { footnotehyper-with-footnote-package } }
8972         { \usepackage { footnotehyper } }
8973     }
8974     \bool_set_true:N \g_@@_footnote_bool
8975 }

```

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.

```

8976 \bool_new:N \l_@@_underscore_loaded_bool
8977 \IfPackageLoadedTF { underscore }
8978 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8979 { }

8980 \hook_gput_code:nnn { begindocument } { . }
8981 {
8982   \bool_if:NF \l_@@_underscore_loaded_bool
8983   {
8984     \IfPackageLoadedTF { underscore }
8985     { \@@_error:n { underscore~after~nicematrix } }
8986     { }
8987   }
8988 }
```

40 Error messages of the package

```

8989 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
8990 { \str_const:Nn \c_@@_available_keys_str { } }
8991 {
8992   \str_const:Nn \c_@@_available_keys_str
8993   { For~a~list~of~the~available~keys,~type~H~<return>. }
8994 }

8995 \seq_new:N \g_@@_types_of_matrix_seq
8996 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8997 {
8998   NiceMatrix ,
8999   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9000 }
9001 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9002 { \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`.

```

9003 \cs_new_protected:Npn \@@_error_too_much_cols:
9004 {
9005   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9006   {
9007     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9008     { \@@_fatal:n { too~much~cols~for~matrix } }
9009     {
9010       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9011       { \@@_fatal:n { too~much~cols~for~matrix } }
9012       {
9013         \bool_if:NF \l_@@_last_col_without_value_bool
```

```

9014         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9015     }
9016 }
9017 }
9018 { \@@_fatal:nn { too-much-cols-for-array } }
9019 }

The following command must not be protected since it's used in an error message.
9020 \cs_new:Npn \@@_message_hdotsfor:
9021 {
9022     \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
9023     { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9024 }
9025 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9026 {
9027     Incompatible~options.\\
9028     You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9029     The~output~will~not~be~reliable.
9030 }
9031 \@@_msg_new:nn { negative~weight }
9032 {
9033     Negative~weight.\\
9034     The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9035     the~value~'\int_use:N \l_@@_weight_int'.\\
9036     The~absolute~value~will~be~used.
9037 }
9038 \@@_msg_new:nn { last~col~not~used }
9039 {
9040     Column~not~used.\\
9041     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9042     in~your~\@@_full_name_env:~.~However,~you~can~go~on.
9043 }
9044 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9045 {
9046     Too-much~columns.\\
9047     In~the~row~\int_eval:n { \c@iRow },~
9048     you~try~to~use~more~columns~
9049     than~allowed~by~your~\@@_full_name_env:~\@@_message_hdotsfor:\\
9050     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9051     (plus~the~exterior~columns).~This~error~is~fatal.
9052 }
9053 \@@_msg_new:nn { too-much-cols-for-matrix }
9054 {
9055     Too-much~columns.\\
9056     In~the~row~\int_eval:n { \c@iRow },~
9057     you~try~to~use~more~columns~than~allowed~by~your~
9058     \@@_full_name_env:~\@@_message_hdotsfor:\\ Recall~that~the~maximal~
9059     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9060     columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9061     Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9062     \token_to_str:N \setcounter\ to~change~that~value).~
9063     This~error~is~fatal.
9064 }
9065 \@@_msg_new:nn { too-much-cols-for-array }
9066 {
9067     Too-much~columns.\\
9068     In~the~row~\int_eval:n { \c@iRow },~
9069     ~you~try~to~use~more~columns~than~allowed~by~your~
9070     \@@_full_name_env:~\@@_message_hdotsfor:\\ The~maximal~number~of~columns~is~
9071     \int_use:N \g_@@_static_num_of_col_int\\
9072     ~(plus~the~potential~exterior~ones).

```

```

9073     This-error-is-fatal.
9074 }

9075 \@@_msg_new:nn { columns-not-used }
9076 {
9077     Columns-not-used.\\
9078     The-preamble-of-your-\@@_full_name_env:\ announces-\int_use:N
9079     \g_@@_static_num_of_col_int\ columns-but-you-use-only-\int_use:N \c@jCol.\\
9080     The-columns-you-did-not-used-won't-be-created.\\
9081     You-won't-have-similar-error-till-the-end-of-the-document.
9082 }

9083 \@@_msg_new:nn { in-first-col }
9084 {
9085     Erroneous-use.\\
9086     You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
9087     That-command-will-be-ignored.
9088 }

9089 \@@_msg_new:nn { in-last-col }
9090 {
9091     Erroneous-use.\\
9092     You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
9093     That-command-will-be-ignored.
9094 }

9095 \@@_msg_new:nn { in-first-row }
9096 {
9097     Erroneous-use.\\
9098     You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
9099     That-command-will-be-ignored.
9100 }

9101 \@@_msg_new:nn { in-last-row }
9102 {
9103     You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
9104     That-command-will-be-ignored.
9105 }

9106 \@@_msg_new:nn { caption-outside-float }
9107 {
9108     Key-caption-forbidden.\\
9109     You-can't-use-the-key~'caption'~because-you-are-not-in-a-floating~
9110     environment.~This-key-will-be-ignored.
9111 }

9112 \@@_msg_new:nn { short-caption-without-caption }
9113 {
9114     You-should-not-use-the-key~'short-caption'~without~'caption'.~
9115     However,~your~'short-caption'~will-be-used-as~'caption'.
9116 }

9117 \@@_msg_new:nn { double-closing-delimiter }
9118 {
9119     Double-delimiter.\\
9120     You-can't-put-a-second-closing-delimiter~"#1"~just-after-a-first-closing~
9121     delimiter.~This-delimiter-will-be-ignored.
9122 }

9123 \@@_msg_new:nn { delimiter-after-opening }
9124 {
9125     Double-delimiter.\\
9126     You-can't-put-a-second-delimiter~"#1"~just-after-a-first-opening~
9127     delimiter.~That-delimiter-will-be-ignored.
9128 }

9129 \@@_msg_new:nn { bad-option-for-line-style }
9130 {
9131     Bad-line-style.\\

```

```

9132     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9133     is~'standard'.~That~key~will~be~ignored.
9134 }

9135 \@@_msg_new:nn { Identical~notes~in~caption }
9136 {
9137     Identical~tabular~notes.\\
9138     You~can't~put~several~notes~with~the~same~content~in~
9139     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9140     If~you~go~on,~the~output~will~probably~be~erroneous.
9141 }

9142 \@@_msg_new:nn { tabularnote~below~the~tabular }
9143 {
9144     \token_to_str:N \tabularnote\ forbidden\\
9145     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9146     of~your~tabular~because~the~caption~will~be~composed~below~
9147     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9148     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9149     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9150     no~similar~error~will~raised~in~this~document.
9151 }

9152 \@@_msg_new:nn { Unknown~key~for~rules }
9153 {
9154     Unknown~key.\\
9155     There~is~only~two~keys~available~here:~width~and~color.\\
9156     Your~key~'\l_keys_key_str'~will~be~ignored.
9157 }

9158 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9159 {
9160     Unknown~key.\\
9161     There~is~only~two~keys~available~here:~
9162     'empty'~and~'not-empty'.\\
9163     Your~key~'\l_keys_key_str'~will~be~ignored.
9164 }

9165 \@@_msg_new:nn { Unknown~key~for~rotate }
9166 {
9167     Unknown~key.\\
9168     The~only~key~available~here~is~'c'.\\
9169     Your~key~'\l_keys_key_str'~will~be~ignored.
9170 }

9171 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9172 {
9173     Unknown~key.\\
9174     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9175     It~you~go~on,~you~will~probably~have~other~errors. \\
9176     \c_@@_available_keys_str
9177 }
9178 {
9179     The~available~keys~are~(in~alphabetic~order):~
9180     ccommand,~
9181     color,~
9182     command,~
9183     dotted,~
9184     letter,~
9185     multiplicity,~
9186     sep-color,~
9187     tikz,~and~total-width.
9188 }

9189 \@@_msg_new:nnn { Unknown~key~for~xdots }
9190 {
9191     Unknown~key.\\
9192     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\

```

```

9193     \c_@@_available_keys_str
9194 }
9195 {
9196     The~available~keys~are~(in~alphabetic~order):~
9197     'color',~
9198     'horizontal~labels',~
9199     'inter',~
9200     'line~style',~
9201     'radius',~
9202     'shorten',~
9203     'shorten~end'~and~'shorten~start'.
9204 }
9205 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9206 {
9207     Unknown~key.\\
9208     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
9209     (and~you~try~to~use~'\l_keys_key_str')\\
9210     That~key~will~be~ignored.
9211 }
9212 \@@_msg_new:nn { label~without~caption }
9213 {
9214     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9215     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9216 }
9217 \@@_msg_new:nn { W~warning }
9218 {
9219     Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
9220     (row~\int_use:N \c@iRow).
9221 }
9222 \@@_msg_new:nn { Construct~too~large }
9223 {
9224     Construct~too~large.\\
9225     Your~command~\token_to_str:N #1
9226     can't~be~drawn~because~your~matrix~is~too~small.\\
9227     That~command~will~be~ignored.
9228 }
9229 \@@_msg_new:nn { underscore~after~nicematrix }
9230 {
9231     Problem~with~'underscore'.\\
9232     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9233     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9234     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9235 }
9236 \@@_msg_new:nn { ampersand~in~light~syntax }
9237 {
9238     Ampersand~forbidden.\\
9239     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9240     ~the~key~'light~syntax'~is~in~force.~This~error~is~fatal.
9241 }
9242 \@@_msg_new:nn { double~backslash~in~light~syntax }
9243 {
9244     Double~backslash~forbidden.\\
9245     You~can't~use~\token_to_str:N
9246     \\~to~separate~rows~because~the~key~'light~syntax'~
9247     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9248     (set~by~the~key~'end~of~row').~This~error~is~fatal.
9249 }
9250 \@@_msg_new:nn { hlines~with~color }
9251 {
9252     Incompatible~keys.\\

```



```

9253     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9254     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9255     Maybe~it~will~possible~in~future~version.\\
9256     Your~key~will~be~discarded.
9257 }
9258 \@@_msg_new:nn { bad-value-for-baseline }
9259 {
9260     Bad~value~for~baseline.\\
9261     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9262     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9263     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9264     the~form~'line-i'.\\
9265     A~value~of~1~will~be~used.
9266 }
9267 \@@_msg_new:nn { detection-of-empty-cells }
9268 {
9269     Problem~with~'not-empty'\\
9270     For~technical~reasons,~you~must~activate~
9271     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9272     in~order~to~use~the~key~'\l_keys_key_str'.\\
9273     That~key~will~be~ignored.
9274 }
9275 \@@_msg_new:nn { siunitx-not-loaded }
9276 {
9277     siunitx~not~loaded\\
9278     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9279     That~error~is~fatal.
9280 }
9281 \@@_msg_new:nn { ragged2e-not-loaded }
9282 {
9283     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9284     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9285     \l_keys_key_str'~will~be~used~instead.
9286 }
9287 \@@_msg_new:nn { Invalid-name }
9288 {
9289     Invalid~name.\\
9290     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9291     \SubMatrix\ of~your~\@@_full_name_env:.\\
9292     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9293     This~key~will~be~ignored.
9294 }
9295 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9296 {
9297     Wrong~line.\\
9298     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9299     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9300     number~is~not~valid.~It~will~be~ignored.
9301 }
9302 \@@_msg_new:nn { Impossible-delimiter }
9303 {
9304     Impossible~delimiter.\\
9305     It's~impossible~to~draw~the~#1~delimiter~of~your~
9306     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9307     in~that~column.
9308     \bool_if:NT \l_@@_submatrix_slim_bool
9309     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9310     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9311 }
9312 \@@_msg_new:nnn { width-without-X-columns }

```

```

9313 {
9314     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9315     That~key~will~be~ignored.
9316 }
9317 {
9318     This~message~is~the~message~'width~without~X~columns'~
9319     of~the~module~'nicematrix'.~
9320     The~experimented~users~can~disable~that~message~with~
9321     \token_to_str:N \msg_redirect_name:nnn.\\
9322 }
9323
9324 \@@_msg_new:nn { key~multiplicity~with~dotted }
9325 {
9326     Incompatible~keys. \\
9327     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9328     in~a~'custom-line'.~They~are~incompatible. \\
9329     The~key~'multiplicity'~will~be~discarded.
9330 }
9331 \@@_msg_new:nn { empty~environment }
9332 {
9333     Empty~environment.\\
9334     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9335 }
9336 \@@_msg_new:nn { No~letter~and~no~command }
9337 {
9338     Erroneous~use.\\
9339     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9340     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9341     '~ccommand'~(to~draw~horizontal~rules).\\
9342     However,~you~can~go~on.
9343 }
9344 \@@_msg_new:nn { Forbidden~letter }
9345 {
9346     Forbidden~letter.\\
9347     You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9348     It~will~be~ignored.
9349 }
9350 \@@_msg_new:nn { Several~letters }
9351 {
9352     Wrong~name.\\
9353     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9354     have~used~'\l_@@_letter_str').\\
9355     It~will~be~ignored.
9356 }
9357 \@@_msg_new:nn { Delimiter~with~small }
9358 {
9359     Delimiter~forbidden.\\
9360     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9361     because~the~key~'small'~is~in~force.\\
9362     This~error~is~fatal.
9363 }
9364 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9365 {
9366     Unknown~cell.\\
9367     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9368     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9369     can't~be~executed~because~a~cell~doesn't~exist.\\
9370     This~command~\token_to_str:N \line\ will~be~ignored.
9371 }
9372 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }

```

```

9373 {
9374   Duplicate-name.\
9375   The-name~'#1'~is~already-used-for-a~\token_to_str:N \SubMatrix\
9376   in~this~\@@_full_name_env:.\
9377   This-key-will-be-ignored.\
9378   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9379   { For-a-list-of-the-names-already-used,~type-H~<return>. }
9380 }
9381 {
9382   The-names-already-defined-in-this~\@@_full_name_env:\ are:~
9383   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9384 }
9385 \@@_msg_new:nn { r~or~l~with~preamble }
9386 {
9387   Erroneous-use.\
9388   You-can't-use-the-key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9389   You-must-specify-the-alignment-of~your~columns~with~the~preamble~of~
9390   your~\@@_full_name_env:.\
9391   This-key-will-be-ignored.
9392 }
9393 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9394 {
9395   Erroneous-use.\
9396   You-can't-use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9397   the~array.~This~error~is~fatal.
9398 }
9399 \@@_msg_new:nn { bad~corner }
9400 {
9401   Bad-corner.\
9402   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9403   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
9404   This~specification~of~corner~will~be~ignored.
9405 }
9406 \@@_msg_new:nn { bad~border }
9407 {
9408   Bad-border.\
9409   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9410   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9411   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9412   also~use~the~key~'tikz'
9413   \IfPackageLoadedTF { tikz }
9414   { }
9415   {~if~you~load~the~LaTeX~package~'tikz'}).\
9416   This~specification~of~border~will~be~ignored.
9417 }
9418 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9419 {
9420   TikZ~not~loaded.\
9421   You-can't-use~\token_to_str:N \TikzEveryCell\
9422   because~you~have~not~loaded~tikz.~
9423   This~command~will~be~ignored.
9424 }
9425 \@@_msg_new:nn { tikz~key~without~tikz }
9426 {
9427   TikZ~not~loaded.\
9428   You-can't-use~the~key~'tikz'~for~the~command~'\token_to_str:N
9429   \Block'~because~you~have~not~loaded~tikz.~
9430   This~key~will~be~ignored.
9431 }
9432 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9433 {

```

```

9434 Erroneous-use.\\
9435 In-the~\@@_full_name_env:,~you-must-use-the-key~
9436 'last-col'~without-value.\\
9437 However,~you-can-go-on~for~this~time~
9438 (the-value~'\l_keys_value_tl'~will-be-ignored).
9439 }

9440 \@@_msg_new:nn { last-col-non-empty-for~NiceMatrixOptions }
9441 {
9442 Erroneous-use.\\
9443 In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9444 'last-col'~without-value.\\
9445 However,~you-can-go-on~for~this~time~
9446 (the-value~'\l_keys_value_tl'~will-be-ignored).
9447 }

9448 \@@_msg_new:nn { Block-too-large-1 }
9449 {
9450 Block-too-large.\\
9451 You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9452 too~small~for~that~block. \\
9453 This~block~and~maybe~others~will~be~ignored.
9454 }

9455 \@@_msg_new:nn { Block-too-large-2 }
9456 {
9457 Block-too-large.\\
9458 The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9459 \g_@@_static_num_of_col_int\
9460 columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9461 specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9462 (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\\
9463 This~block~and~maybe~others~will~be~ignored.
9464 }

9465 \@@_msg_new:nn { unknown-column-type }
9466 {
9467 Bad-column-type.\\
9468 The~column-type~'#1'~in~your~\@@_full_name_env:\
9469 is~unknown. \\
9470 This~error~is~fatal.
9471 }

9472 \@@_msg_new:nn { unknown-column-type-S }
9473 {
9474 Bad-column-type.\\
9475 The~column-type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9476 If~you~want~to~use~the~column-type~'S'~of~siunitx,~you~should~
9477 load~that~package. \\
9478 This~error~is~fatal.
9479 }

9480 \@@_msg_new:nn { tabularnote-forbidden }
9481 {
9482 Forbidden-command.\\
9483 You~can't~use~the~command~\token_to_str:N\tabularnote\
9484 ~here.~This~command~is~available~only~in~
9485 \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9486 the~argument~of~a~command~\token_to_str:N \caption\ included~
9487 in~an~environment~{table}. \\
9488 This~command~will~be~ignored.
9489 }

9490 \@@_msg_new:nn { borders-forbidden }
9491 {
9492 Forbidden-key.\\
9493 You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9494 because~the~option~'rounded-corners'~

```

```

9495     is~in~force~with~a~non~zero~value.\\
9496     This~key~will~be~ignored.
9497 }

9498 \@@_msg_new:nn { bottomrule~without~booktabs }
9499 {
9500     booktabs~not~loaded.\\
9501     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9502     loaded~'booktabs'.\\
9503     This~key~will~be~ignored.
9504 }

9505 \@@_msg_new:nn { enumitem~not~loaded }
9506 {
9507     enumitem~not~loaded.\\
9508     You~can't~use~the~command~\token_to_str:N\tabularnote\
9509     ~because~you~haven't~loaded~'enumitem'.\\
9510     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9511     ignored~in~the~document.
9512 }

9513 \@@_msg_new:nn { tikz~without~tikz }
9514 {
9515     Tikz~not~loaded.\\
9516     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9517     loaded.~If~you~go~on,~that~key~will~be~ignored.
9518 }

9519 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9520 {
9521     Tikz~not~loaded.\\
9522     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9523     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9524     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9525     use~that~custom~line.
9526 }

9527 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9528 {
9529     Tikz~not~loaded.\\
9530     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9531     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9532     That~key~will~be~ignored.
9533 }

9534 \@@_msg_new:nn { without~color~inside }
9535 {
9536     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9537     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9538     outside~\token_to_str:N \CodeBefore,~you~
9539     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\\
9540     You~can~go~on~but~you~may~need~more~compilations.
9541 }

9542 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9543 {
9544     Erroneous~use.\\
9545     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9546     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9547     The~key~'color'~will~be~discarded.
9548 }

9549 \@@_msg_new:nn { Wrong~last~row }
9550 {
9551     Wrong~number.\\
9552     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9553     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9554     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~

```

```

9555     last~row~.~You~can~avoid~this~problem~by~using~'last~row'~
9556     without~value~(more~compilations~might~be~necessary).
9557 }

9558 \@@_msg_new:nn { Yet~in~env }
9559 {
9560     Nested~environments.\\
9561     Environments~of~nicematrix~can't~be~nested.\\
9562     This~error~is~fatal.
9563 }

9564 \@@_msg_new:nn { Outside~math~mode }
9565 {
9566     Outside~math~mode.\\
9567     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9568     (and~not~in~\token_to_str:N \vcenter).\\
9569     This~error~is~fatal.
9570 }

9571 \@@_msg_new:nn { One~letter~allowed }
9572 {
9573     Bad~name.\\
9574     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9575     It~will~be~ignored.
9576 }

9577 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9578 {
9579     Environment~{TabularNote}~forbidden.\\
9580     You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9581     but~*before*~the~\token_to_str:N \CodeAfter.\\
9582     This~environment~{TabularNote}~will~be~ignored.
9583 }

9584 \@@_msg_new:nn { varwidth~not~loaded }
9585 {
9586     varwidth~not~loaded.\\
9587     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9588     loaded.\\
9589     Your~column~will~behave~like~'p'.
9590 }

9591 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9592 {
9593     Unknow~key.\\
9594     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9595     \c_@@_available_keys_str
9596 }
9597 {
9598     The~available~keys~are~(in~alphabetic~order):~
9599     color,~
9600     dotted,~
9601     multiplicity,~
9602     sep-color,~
9603     tikz,~and~total-width.
9604 }
9605

9606 \@@_msg_new:nnn { Unknown~key~for~Block }
9607 {
9608     Unknown~key.\\
9609     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9610     \Block.\\ It~will~be~ignored. \\
9611     \c_@@_available_keys_str
9612 }
9613 {
9614     The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9615     hlines,~hvlines,~l,~line-width,~name,~opacity,~rounded-corners,~r,~

```

```

9616     respect-arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9617 }
9618 \@@_msg_new:nnn { Unknown~key~for~Brace }
9619 {
9620     Unknown~key.\\
9621     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9622     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9623     It~will~be~ignored. \\
9624     \c_@@_available_keys_str
9625 }
9626 {
9627     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9628     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9629     right~shorten)~and~yshift.
9630 }
9631 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9632 {
9633     Unknown~key.\\
9634     The~key~'\l_keys_key_str'~is~unknown.\\
9635     It~will~be~ignored. \\
9636     \c_@@_available_keys_str
9637 }
9638 {
9639     The~available~keys~are~(in~alphabetic~order):~
9640     delimiters/color,~
9641     rules~(with~the~subkeys~'color'~and~'width'),~
9642     sub-matrix~(several~subkeys)~
9643     and~xdots~(several~subkeys).~
9644     The~latter~is~for~the~command~\token_to_str:N \line.
9645 }
9646 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9647 {
9648     Unknown~key.\\
9649     The~key~'\l_keys_key_str'~is~unknown.\\
9650     It~will~be~ignored. \\
9651     \c_@@_available_keys_str
9652 }
9653 {
9654     The~available~keys~are~(in~alphabetic~order):~
9655     create-cell-nodes,~
9656     delimiters/color~and~
9657     sub-matrix~(several~subkeys).
9658 }
9659 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9660 {
9661     Unknown~key.\\
9662     The~key~'\l_keys_key_str'~is~unknown.\\
9663     That~key~will~be~ignored. \\
9664     \c_@@_available_keys_str
9665 }
9666 {
9667     The~available~keys~are~(in~alphabetic~order):~
9668     'delimiters/color',~
9669     'extra-height',~
9670     'hlines',~
9671     'hvlines',~
9672     'left-xshift',~
9673     'name',~
9674     'right-xshift',~
9675     'rules'~(with~the~subkeys~'color'~and~'width'),~
9676     'slim',~
9677     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~

```

```

9678     and~'right-xshift').\\
9679 }
9680 \@@_msg_new:nnn { Unknown~key~for~notes }
9681 {
9682     Unknown~key.\\
9683     The~key~'\l_keys_key_str'~is~unknown.\\
9684     That~key~will~be~ignored. \\
9685     \c_@@_available_keys_str
9686 }
9687 {
9688     The~available~keys~are~(in~alphabetic~order):~
9689     bottomrule,~
9690     code~after,~
9691     code~before,~
9692     detect~duplicates,~
9693     enumitem~keys,~
9694     enumitem~keys~para,~
9695     para,~
9696     label~in~list,~
9697     label~in~tabular~and~
9698     style.
9699 }
9700 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9701 {
9702     Unknown~key.\\
9703     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9704     \token_to_str:N \RowStyle. \\
9705     That~key~will~be~ignored. \\
9706     \c_@@_available_keys_str
9707 }
9708 {
9709     The~available~keys~are~(in~alphabetic~order):~
9710     'bold',~
9711     'cell-space-top-limit',~
9712     'cell-space-bottom-limit',~
9713     'cell-space-limits',~
9714     'color',~
9715     'nb-rows'~and~
9716     'rowcolor'.
9717 }
9718 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9719 {
9720     Unknown~key.\\
9721     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9722     \token_to_str:N \NiceMatrixOptions. \\
9723     That~key~will~be~ignored. \\
9724     \c_@@_available_keys_str
9725 }
9726 {
9727     The~available~keys~are~(in~alphabetic~order):~
9728     allow~duplicate~names,~
9729     caption~above,~
9730     cell-space-bottom-limit,~
9731     cell-space-limits,~
9732     cell-space-top-limit,~
9733     code~for~first~col,~
9734     code~for~first~row,~
9735     code~for~last~col,~
9736     code~for~last~row,~
9737     corners,~
9738     custom~key,~
9739     create~extra~nodes,~
9740     create~medium~nodes,~

```



```

9741 create-large-nodes,~
9742 delimiters~(several~subkeys),~
9743 end-of-row,~
9744 first-col,~
9745 first-row,~
9746 hlines,~
9747 hvlines,~
9748 hvlines-except-borders,~
9749 last-col,~
9750 last-row,~
9751 left-margin,~
9752 light-syntax,~
9753 matrix/columns-type,~
9754 notes~(several~subkeys),~
9755 nullify-dots,~
9756 pgf-node-code,~
9757 renew-dots,~
9758 renew-matrix,~
9759 respect-arraystretch,~
9760 rounded-corners,~
9761 right-margin,~
9762 rules~(with~the~subkeys~'color'~and~'width'),~
9763 small,~
9764 sub-matrix~(several~subkeys),~
9765 vlines,~
9766 xdots~(several~subkeys).
9767 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

9768 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9769 {
9770   Unknown~key.\\
9771   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9772   \{NiceArray\}. \\
9773   That~key~will~be~ignored. \\
9774   \c_@@_available_keys_str
9775 }
9776 {
9777   The~available~keys~are~(in~alphabetic~order):~
9778   b,~
9779   baseline,~
9780   c,~
9781   cell-space-bottom-limit,~
9782   cell-space-limits,~
9783   cell-space-top-limit,~
9784   code-after,~
9785   code-for-first-col,~
9786   code-for-first-row,~
9787   code-for-last-col,~
9788   code-for-last-row,~
9789   color-inside,~
9790   columns-width,~
9791   corners,~
9792   create-extra-nodes,~
9793   create-medium-nodes,~
9794   create-large-nodes,~
9795   extra-left-margin,~
9796   extra-right-margin,~
9797   first-col,~
9798   first-row,~
9799   hlines,~
9800   hvlines,~
9801   hvlines-except-borders,~

```

```

9802     last-col,~
9803     last-row,~
9804     left-margin,~
9805     light-syntax,~
9806     name,~
9807     nullify-dots,~
9808     pgf-node-code,~
9809     renew-dots,~
9810     respect-arraystretch,~
9811     right-margin,~
9812     rounded-corners,~
9813     rules~(with~the~subkeys~'color'~and~'width'),~
9814     small,~
9815     t,~
9816     vlines,~
9817     xdots/color,~
9818     xdots/shorten-start,~
9819     xdots/shorten-end,~
9820     xdots/shorten~and~
9821     xdots/line-style.
9822 }

```

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`).

```

9823 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9824 {
9825     Unknown~key.\\
9826     The~key~'\l_keys_key_str'~is~unknown~for~the~
9827     \@@_full_name_env:. \\
9828     That~key~will~be~ignored. \\
9829     \c_@@_available_keys_str
9830 }
9831 {
9832     The~available~keys~are~(in~alphabetic~order):~
9833     b,~
9834     baseline,~
9835     c,~
9836     cell-space-bottom-limit,~
9837     cell-space-limits,~
9838     cell-space-top-limit,~
9839     code-after,~
9840     code-for-first-col,~
9841     code-for-first-row,~
9842     code-for-last-col,~
9843     code-for-last-row,~
9844     color-inside,~
9845     columns-type,~
9846     columns-width,~
9847     corners,~
9848     create-extra-nodes,~
9849     create-medium-nodes,~
9850     create-large-nodes,~
9851     extra-left-margin,~
9852     extra-right-margin,~
9853     first-col,~
9854     first-row,~
9855     hlines,~
9856     hvlines,~
9857     hvlines-except-borders,~
9858     l,~
9859     last-col,~
9860     last-row,~
9861     left-margin,~
9862     light-syntax,~

```

```

9863     name,~
9864     nullify-dots,~
9865     pgf-node-code,~
9866     r,~
9867     renew-dots,~
9868     respect-arraystretch,~
9869     right-margin,~
9870     rounded-corners,~
9871     rules~(with~the~subkeys~'color'~and~'width'),~
9872     small,~
9873     t,~
9874     vlines,~
9875     xdots/color,~
9876     xdots/shorten-start,~
9877     xdots/shorten-end,~
9878     xdots/shorten-and~
9879     xdots/line-style.
9880 }
9881 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9882 {
9883     Unknown~key.\\
9884     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9885     \{NiceTabular\}. \\
9886     That~key~will~be~ignored. \\
9887     \c_@@_available_keys_str
9888 }
9889 {
9890     The~available~keys~are~(in~alphabetic~order):~
9891     b,~
9892     baseline,~
9893     c,~
9894     caption,~
9895     cell-space-bottom-limit,~
9896     cell-space-limits,~
9897     cell-space-top-limit,~
9898     code-after,~
9899     code-for-first-col,~
9900     code-for-first-row,~
9901     code-for-last-col,~
9902     code-for-last-row,~
9903     color-inside,~
9904     columns-width,~
9905     corners,~
9906     custom-line,~
9907     create-extra-nodes,~
9908     create-medium-nodes,~
9909     create-large-nodes,~
9910     extra-left-margin,~
9911     extra-right-margin,~
9912     first-col,~
9913     first-row,~
9914     hlines,~
9915     hvlines,~
9916     hvlines-except-borders,~
9917     label,~
9918     last-col,~
9919     last-row,~
9920     left-margin,~
9921     light-syntax,~
9922     name,~
9923     notes~(several~subkeys),~
9924     nullify-dots,~
9925     pgf-node-code,~

```

```

9926   renew-dots,~
9927   respect-arraystretch,~
9928   right-margin,~
9929   rounded-corners,~
9930   rules~(with~the~subkeys~'color'~and~'width'),~
9931   short-caption,~
9932   t,~
9933   tabularnote,~
9934   vlines,~
9935   xdots/color,~
9936   xdots/shorten-start,~
9937   xdots/shorten-end,~
9938   xdots/shorten~and~
9939   xdots/line-style.
9940 }

9941 \@@_msg_new:nnn { Duplicate-name }
9942 {
9943   Duplicate-name.\\
9944   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9945   the~same~environment~name~twice.~You~can~go~on,~but,~
9946   maybe,~you~will~have~incorrect~results~especially~
9947   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9948   message~again,~use~the~key~'allow-duplicate-names'~in~
9949   '\token_to_str:N \NiceMatrixOptions'.\\
9950   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9951     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9952 }
9953 {
9954   The~names~already~defined~in~this~document~are:~
9955   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9956 }

9957 \@@_msg_new:nn { Option-auto-for-columns-width }
9958 {
9959   Erroneous~use.\\
9960   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9961   That~key~will~be~ignored.
9962 }

9963 \@@_msg_new:nn { NiceTabularX-without-X }
9964 {
9965   NiceTabularX~without~X.\\
9966   You~should~not~use~{NiceTabularX}~without~X~columns.\\
9967   However,~you~can~go~on.
9968 }

9969 \@@_msg_new:nn { Preamble-forgotten }
9970 {
9971   Preamble~forgotten.\\
9972   You~have~probably~forgotten~the~preamble~of~your~
9973   \@@_full_name_env:. \\
9974   This~error~is~fatal.
9975 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
3	Collecting options	4
4	Technical definitions	4
5	Parameters	10
6	The command <code>\tabularnote</code>	20
7	Command for creation of rectangle nodes	25
8	The options	26
9	Important code used by <code>{NiceArrayWithDelims}</code>	36
10	The <code>\CodeBefore</code>	49
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	58
13	The redefinition of <code>\multicolumn</code>	72
14	The environment <code>{NiceMatrix}</code> and its variants	90
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	91
16	After the construction of the array	92
17	We draw the dotted lines	98
18	The actual instructions for drawing the dotted lines with Tikz	112
19	User commands available in the new environments	117
20	The command <code>\line</code> accessible in code-after	123
21	The command <code>\RowStyle</code>	125
22	Colors of cells, rows and columns	128
23	The vertical and horizontal rules	139
24	The empty corners	153
25	The environment <code>{NiceMatrixBlock}</code>	156
26	The extra nodes	157
27	The blocks	161
28	How to draw the dotted lines transparently	181
29	Automatic arrays	182
30	The redefinition of the command <code>\dotfill</code>	183

31	The command <code>\diagbox</code>	183
32	The keyword <code>\CodeAfter</code>	185
33	The delimiters in the preamble	186
34	The command <code>\SubMatrix</code>	187
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	195
36	The command <code>TikzEveryCell</code>	198
37	The command <code>\ShowCellNames</code>	200
38	We process the options at package loading	202
39	About the package underscore	204
40	Error messages of the package	204