

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 24, 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 6.25 of `nicematrix`, at the date of 2023/10/24.

```

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 \@one \fi
323 \def\the@toks{\the\toks}
```

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

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

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

```
326 \bool_if:NTF \c_@@_revtex_bool
327 { \cs_new_protected:Npn \@_redefine_mkpream: { } }
328 {
329 \cs_new_protected:Npn \@_redefine_mkpream:
330 { \cs_set_eq:NN \@mkpream \@_mkpream: }
331 }
```

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

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

```

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

5 Parameters

The following counter will count the environments `{NiceArray}`. The value of this counter will be used to prefix the names of the Tikz nodes created in the array.

```

350 \int_new:N \g_@@_env_int
```

The following command is only a syntactic shortcut. It must *not* be protected (it will be used in names of PGF nodes).

```

351 \cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }
```

The command `\NiceMatrixLastEnv` is not used by the package `nicematrix`. It's only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it's meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name). This command *must* be expandable since it will be used in pgf nodes.

```

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

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

```

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

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

```

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

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

```

357 \bool_new:N \g_@@_delims_bool
358 \bool_gset_true:N \g_@@_delims_bool
```

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

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

```

359 \bool_new:N \l_@@_preamble_bool
360 \bool_set_true:N \l_@@_preamble_bool
```

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

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

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

```
362 \int_new:N \g_@@_NiceMatrixBlock_int
```

It's possible to put tabular notes (with `\tabularnote`) in the caption if that caption is composed *above* the tabular. In such case, we will count in `\g_@@_notes_caption_int` the number of uses of the command `\tabularnote` *without optional argument* in that caption.

```
363 \int_new:N \g_@@_notes_caption_int
```

The dimension `\l_@@_columns_width_dim` will be used when the options specify that all the columns must have the same width (but, if the key `columns-width` is used with the special value `auto`, the boolean `\l_@@_auto_columns_width_bool` also will be raised).

```
364 \dim_new:N \l_@@_columns_width_dim
```

The dimension `\l_@@_col_width_dim` will be available in each cell which belongs to a column of fixed width: `w{...}{...}`, `W{...}{...}`, `p{}`, `m{}`, `b{}` but also `X` (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands `\Block`. A non positive value means that the column has no fixed width (it's a column of type `c`, `r`, `l`, etc.).

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

```
366 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

The following counters will be used to count the numbers of rows and columns of the array.

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

```
368 \int_new:N \g_@@_col_total_int
```

The following parameter will be used by `\@@_create_row_node:` to avoid to create the same row-node twice (at the end of the array).

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

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

```
370 \int_new:N \l_@@_key_nb_rows_int
```

The following token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are `r`, `l`, `c` and `j`. For example, a column `p[1]{3cm}` will provide the value `l` for all the cells of the column.

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

```
372 \str_set:Nn \l_@@_hpos_cell_str { c }
```

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

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

Idem for the mono-row blocks.

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

```
375 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

```
377 \dim_new:N \l_@@_width_dim
```

The sequence `\g_@@_names_seq` will be the list of all the names of environments used (via the option `name`) in the document: two environments must not have the same name. However, it's possible to use the option `allow-duplicate-names`.

```
378 \seq_new:N \g_@@_names_seq
```

We want to know whether we are in an environment of `nicematrix` because we will raise an error if the user tries to use nested environments.

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

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

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

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

```
382 \dim_new:N \l_@@_tabular_width_dim
```

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

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

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

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

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

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

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

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

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

```
387 \bool_new:N \l_@@_X_column_bool
388 \bool_new:N \g_@@_caption_finished_bool
```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`).

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

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

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

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

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

```

392 \tl_new:N \g_@@_left_delim_tl
393 \tl_new:N \g_@@_right_delim_tl

```

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

```

394 \tl_new:N \g_@@_user_preamble_tl

```

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

```

395 \tl_new:N \g_@@_array_preamble_tl

```

For `\multicolumn`.

```

396 \tl_new:N \g_@@_preamble_tl

```

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`.

```

397 \tl_new:N \l_@@_columns_type_tl
398 \tl_set:Nn \l_@@_columns_type_tl { c }

```

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

```

399 \tl_new:N \l_@@_xdots_down_tl
400 \tl_new:N \l_@@_xdots_up_tl
401 \tl_new:N \l_@@_xdots_middle_tl

```

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

```

402 \seq_new:N \g_@@_rowlistcolors_seq

```

```

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

```

The list of the columns where vertical lines in sub-matrices (`vlism`) must be drawn. Of course, the actual value of this sequence will be known after the analyse of the preamble of the array.

```

409 \seq_new:N \g_@@_cols_vlism_seq

```

The following colors will be used to memorize the color of the potential “first col” and the potential “first row”.

```

410 \colorlet { nicematrix-last-col } { . }
411 \colorlet { nicematrix-last-row } { . }

```

The following string is the name of the current environment or the current command of `nicematrix` (despite its name which contains `env`).

```

412 \str_new:N \g_@@_name_env_str

```

The following string will contain the word *command* or *environment* whether we are in a command of `nicematrix` or in an environment of `nicematrix`. The default value is *environment*.

```

413 \tl_new:N \g_@@_com_or_env_str
414 \tl_gset:Nn \g_@@_com_or_env_str { environment }

```

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

```

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

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

```

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

For the key `pgf-node-code`. That code will be used when the nodes of the cells (that is to say the nodes of the form *i-j*) will be created.

```

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

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

```

423
```

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

```

424 \tl_new:N \g_@@_pre_code_before_tl
425 \tl_new:N \g_nicematrix_code_before_tl
```

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

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

```

426 \tl_new:N \g_@@_pre_code_after_tl
427 \tl_new:N \g_nicematrix_code_after_tl
```

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

```

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

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

```

429 \int_new:N \l_@@_old_iRow_int
430 \int_new:N \l_@@_old_jCol_int
```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

The following sequence will contain the names (without backslash) of the commands created by `custom-line` by the key `command` or `ccommand` (commands used by the final user in order to draw horizontal rules).

```

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

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

```

432 \tl_new:N \l_@@_rules_color_tl
```

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as an optional argument between square brackets. The default value, of course, is 1.

```
433 \int_new:N \g_@@_total_X_weight_int
```

If there is at least one X-column in the preamble of the array, the following flag will be raised via the aux file. The length `l_@@_x_columns_dim` will be the width of X-columns of weight 1 (the width of a column of weight n will be that dimension multiplied by n). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
434 \bool_new:N \l_@@_X_columns_aux_bool
435 \dim_new:N \l_@@_X_columns_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
436 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the `col` nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of `col` nodes).

```
437 \bool_new:N \g_@@_row_of_col_done_bool
```

It's possible to use the command `\NotEmpty` to specify explicitly that a cell must be considered as non empty by `nicematrix` (the Tikz nodes are constructed only in the non empty cells).

```
438 \bool_new:N \g_@@_not_empty_cell_bool
```

`\l_@@_code_before_tl` may contain two types of informations:

- A code-before written in the aux file by a previous run. When the aux file is read, this code-before is stored in `\g_@@_code_before_i_tl` (where i is the number of the environment) and, at the beginning of the environment, it will be put in `\l_@@_code_before_tl`.
- The final user can explicitly add material in `\l_@@_code_before_tl` by using the key `code-before` or the keyword `\CodeBefore` (with the keyword `\Body`).

```
439 \tl_new:N \l_@@_code_before_tl
440 \bool_new:N \l_@@_code_before_bool
```

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

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

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

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

The L3 programming layer provides scratch dimensions `\l_tmpa_dim` and `\l_tmpb_dim`. We creates two more in the same spirit.

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

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

```
448 \bool_new:N \g_@@_empty_cell_bool
```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

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

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces:

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

The variable is global because it will be modified in the cells of the array.

```
451 \seq_new:N \g_@@_blocks_seq
```

We also manage a sequence of the *positions* of the blocks. In that sequence, each block is represented by only five components: `{imin}{jmin}{imax}{jmax}{ name}`. A block with the key `hvlines` won’t appear in that sequence (otherwise, the lines in that block would not be drawn!).

```
452 \seq_new:N \g_@@_pos_of_blocks_seq
```

In fact, this sequence will also contain the positions of the cells with a `\diagbox`. The sequence `\g_@@_pos_of_blocks_seq` will be used when we will draw the rules (which respect the blocks).

We will also manage a sequence for the positions of the dotted lines. These dotted lines are created in the array by `\Cdots`, `\Vdots`, `\Ddots`, etc. However, their positions, that is to say, their extremities, will be determined only after the construction of the array. In this sequence, each item contains five components: `{imin}{jmin}{imax}{jmax}{ name}`.

```
453 \seq_new:N \g_@@_pos_of_xdots_seq
```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won’t be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That’s why we keep the information of all that stroken blocks in the following sequence.

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

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

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

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

```
456 \seq_new:N \g_@@_submatrix_names_seq
```

The following flag will be raised if the key `width` is used in an environment `{NiceTabular}` (not in a command `\NiceMatrixOptions`). You use it to raise an error when this key is used while no column `X` is used.

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

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

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

```
459 \seq_new:N \g_@@_multicolumn_sizes_seq
```

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the code-before).

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

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

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

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


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

```
464 \seq_new:N \g_@@_submatrix_seq
```

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble of course and without the potential exterior columns).

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

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

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

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by `nicematrix` when the key `corners` is used).

The following dimension corresponds to the key `rounded-corners` available in an individual environment `{NiceTabular}`. When that key is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```
472 \dim_new:N \l_@@_tab_rounded_corners_dim
```

The following token list correspond to the key `color` of the command `\Block` and also the key `color` of the command `\RowStyle`.

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

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

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

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

```
475 \dim_new:N \l_@@_line_width_dim
```

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

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

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

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

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

```
481 \bool_new:N \l_@@_draw_first_bool
```

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

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

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

```

484 \int_new:N \g_@@_block_box_int

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

```

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

```

492 \bool_new:N \l_@@_hvlines_bool

```

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

```

493 \bool_new:N \l_@@_dotted_bool

```

The following flag will be set to `true` during the composition of a caption specified (by the key `caption`).

```

494 \bool_new:N \l_@@_in_caption_bool

```

Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

• First row

The integer `\l_@@_first_row_int` is the number of the first row of the array. The default value is 1, but, if the option `first-row` is used, the value will be 0.

```

495 \int_new:N \l_@@_first_row_int
496 \int_set:Nn \l_@@_first_row_int 1

```

• First column

The integer `\l_@@_first_col_int` is the number of the first column of the array. The default value is 1, but, if the option `first-col` is used, the value will be 0.

```

497 \int_new:N \l_@@_first_col_int
498 \int_set:Nn \l_@@_first_col_int 1

```

• Last row

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

```

499 \int_new:N \l_@@_last_row_int
500 \int_set:Nn \l_@@_last_row_int { -2 }

```

If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.²

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

Idem for `\l_@@_last_col_without_value_bool`

```
502 \bool_new:N \l_@@_last_col_without_value_bool
```

• Last column

For the potential “last column”, we use an integer. A value of -2 means that there is no last column. A value of -1 means that we are in an environment without preamble (e.g. `{bNiceMatrix}`) and there is a last column but we don’t know its value because the user has used the option `last-col` without value. A value of 0 means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`): in this case, the key was necessary without argument. The command `\NiceMatrixOptions` also sets `\l_@@_last_col_int` to 0 .

```
503 \int_new:N \l_@@_last_col_int
504 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

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

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

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

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

Some utilities

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

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers.

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

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

```

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

```

The following internal parameters are for:

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

```

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

```

6 The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It’s also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that’s mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That’s why:
 - The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.³
 - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each composant of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first composante is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker `\c_novalue_tl`).

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

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

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

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

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

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

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

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

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

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

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

```
549 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }
```

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

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

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

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

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

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

The type of list `tabularnotes` will be used to format the tabular notes at the end of the array in the general case and `tabularnotes*` will be used if the key `para` is in force.

```

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

```

One must remind that we have allowed a `\tabular` in the caption and that caption may also be found in the list of tables (`\listoftables`). We want the command `\tabularnote` be no-op during the composition of that list. That's why we program `\tabularnote` to be no-op excepted in a floating environment or in an environment of `nicematrix`.

```

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

```

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

```

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

```

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

```

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

```

601 {

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

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

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

```

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

```

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

```

636   \hbox_set:Nn \l_tmpa_box
637   {

```

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

```

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

```

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

```

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

```

```

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

```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array`?) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```

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

```

Now the version when the command is used in the key `caption`. The main difficulty is that the argument of the command `\caption` is composed several times. In order to know the number of commands `\tabularnote` in the caption, we will consider that there should not be the same tabular note twice in the caption (in the main tabular, it's possible). Once we have found a tabular note which has yet been encountered, we consider that you are in a new composition of the argument of `\caption`.

```

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

```

Now, we try to detect duplicate notes in the caption. Be careful! We must put `\tl_if_in:NnF` and not `\tl_if_in:NnT`!

```

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

```

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

```

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

```

Now, we know that are in the second composition of the caption since we are reading a tabular note which has yet been read. Now, the value of `\g_@@_notes_caption_int` won't change anymore: it's the number of uses *without optional argument* of the command `\tabularnote` in the caption.

```

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

```

Now, we will compose the label of the footnote (in the caption). Even if we are not in the first composition, we have to compose that label!

```

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

```



```

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

```

7 Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).

#1 is the name of the node which will be created; **#2** and **#3** are the coordinates of one of the corner of the rectangle; **#4** and **#5** are the coordinates of the opposite corner.

```

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

```

The command `\@@_pgf_rect_node:nnn` is a variant of `\@@_pgf_rect_node:nnnnn`: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

```

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

```

```

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

```

8 The options

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

```

742 \tl_new:N \l_@@_caption_tl
743 \tl_new:N \l_@@_short_caption_tl
744 \tl_new:N \l_@@_label_tl

```

The following parameter corresponds to the key `caption-above` of `\NiceMatrixOptions`. When this parameter is `true`, the captions of the environments `{NiceTabular}`, specified with the key `caption` are put above the tabular (and below elsewhere).

```

745 \bool_new:N \l_@@_caption_above_bool

```

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

```

746 \bool_new:N \l_@@_color_inside_bool

```

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

```

747 \bool_new:N \l_@@_standard_cline_bool

```

The following dimensions correspond to the options `cell-space-top-limit` and `co` (these parameters are inspired by the package `cellspace`).

```

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

```

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

```

750 \bool_new:N \l_@@_xdots_h_labels_bool

```

The following dimension is the distance between two dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.45 em but it will be changed if the option `small` is used.

```

751 \dim_new:N \l_@@_xdots_inter_dim
752 \hook_gput_code:nnn { begindocument } { . }
753 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

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

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

```

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

```

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

The following dimension is the radius of the dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.53 pt but it will be changed if the option `small` is used.

```
761 \dim_new:N \l_@@_xdots_radius_dim
762 \hook_gput_code:nnn { begindocument } { . }
763 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

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

The token list `\l_@@_xdots_line_style_tl` corresponds to the option `tikz` of the commands `\Cdots`, `\Ldots`, etc. and of the options `line-style` for the environments and `\NiceMatrixOptions`. The constant `\c_@@_standard_tl` will be used in some tests.

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

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

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

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

```
768 \tl_new:N \l_@@_baseline_tl
769 \tl_set:Nn \l_@@_baseline_tl c
```

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

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

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

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

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

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

```
774 \dim_new:N \l_@@_notes_above_space_dim
775 \hook_gput_code:nnn { begindocument } { . }
776 { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

We use a hook only by security in case `revtex4-1` is used (even though it is obsolete).

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

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

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

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

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

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

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

```
780 \bool_new:N \g_@@_recreate_cell_nodes_bool
```

The string `\l_@@_name_str` will contain the optional name of the environment: this name can be used to access to the Tikz nodes created in the array from outside the environment.

```
781 \str_new:N \l_@@_name_str
```

The boolean `\l_@@_medium_nodes_bool` will be used to indicate whether the “medium nodes” are created in the array. Idem for the “large nodes”.

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

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

The boolean `\l_@@_except_borders_bool` will be raised when the key `hvlines-except-borders` will be used (but that key has also other effects).

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

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

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

```
786 \dim_new:N \l_@@_right_margin_dim
```

The dimensions `\l_@@_extra_left_margin_dim` and `\l_@@_extra_right_margin_dim` correspond to the options `extra-left-margin` and `extra-right-margin`.

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

```
788 \dim_new:N \l_@@_extra_right_margin_dim
```

The token list `\l_@@_end_of_row_tl` corresponds to the option `end-of-row`. It specifies the symbol used to mark the ends of rows when the light syntax is used.

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

```
790 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

The following parameter is for the color the dotted lines drawn by `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` but *not* the dotted lines drawn by `\hdottedline` and “:”.

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

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

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

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

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

```

794 \keys_define:nn { NiceMatrix / xdots }
795 {
796   shorten-start .code:n =
797     \hook_gput_code:nnn { begindocument } { . }
798     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
799   shorten-end .code:n =
800     \hook_gput_code:nnn { begindocument } { . }
801     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
802   shorten-start .value_required:n = true ,
803   shorten-end .value_required:n = true ,
804   shorten .code:n =
805     \hook_gput_code:nnn { begindocument } { . }
806     {
807       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
808       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
809     } ,
810   shorten .value_required:n = true ,
811   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
812   horizontal-labels .default:n = true ,
813   line-style .code:n =
814     {
815       \bool_lazy_or:nnTF
816         { \cs_if_exist_p:N \tikzpicture }
817         { \str_if_eq_p:nn { #1 } { standard } }
818         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
819         { \@@_error:n { bad~option~for~line-style } }
820     } ,
821   line-style .value_required:n = true ,
822   color .tl_set:N = \l_@@_xdots_color_tl ,
823   color .value_required:n = true ,
824   radius .code:n =
825     \hook_gput_code:nnn { begindocument } { . }
826     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
827   radius .value_required:n = true ,
828   inter .code:n =
829     \hook_gput_code:nnn { begindocument } { . }
830     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
831   radius .value_required:n = true ,

```

The options down, up and middle are not documented for the final user because he should use the syntax with \wedge , $_$ and \cdot . 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 } , % modified 2023-08-09
833   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
834   middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

905     \clist_set:Nn \l_@@_vlines_clist { all }
906     \clist_set:Nn \l_@@_hlines_clist { all }
907     \bool_set_true:N \l_@@_hvlines_bool
908     \bool_set_true:N \l_@@_except_borders_bool
909   } ,
910   parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

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

```

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

```

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

```

The options `c`, `t` and `b` of the environment `{NiceArray}` have the same meaning as the option of the classical environment `{array}`.

```

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

```

We test whether we are in the measuring phase of an environment of `amsmath` (always loaded by `nicematrix`) because we want to avoid a fallacious message of duplicate name in this case.

```

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

```



```

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

```

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

```

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

```

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

```

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

```

```

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

```

With the option `renew-matrix`, the environment `{matrix}` of `amsmath` and its variants are redefined to behave like the environment `{NiceMatrix}` and its variants.

```

1080 renew-matrix .code:n = \@@_renew_matrix: ,
1081 renew-matrix .value_forbidden:n = true ,

```

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```

1082 exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,

```

If the option `columns-width` is used, all the columns will have the same width.

In `\NiceMatrixOptions`, the special value `auto` is not available.

```

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

```

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

```

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

```

`\NiceMatrixOptions` is the command of the `nicematrix` package to fix options at the document level. The scope of these specifications is the current TeX group.

```

1100 \NewDocumentCommand \NiceMatrixOptions { m }
1101 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

We finalise the definition of the set of keys “`NiceMatrix / NiceMatrix`”. That set of keys will be used by `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

```

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

```

```

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

```

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

```

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

```

In the environments {NiceArray} and its variants, the option last-col must be used without value because the number of columns of the array is read from the preamble of the array.

```

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

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

```

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

```

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

```

The dimension width will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

```

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

```

```

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

```

The `\CodeAfter` (inserted with the key `code-after` or after the keyword `\CodeAfter`) may always begin with a list of pairs *key=value* between square brackets. Here is the corresponding set of keys. We *must* put the following instructions *after* the :

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

```

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

```

9 Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_cell_begin:w-\@@_cell_end:` will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a `\halign` (via an environment `{array}`).

```

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

```

`\g_@@_cell_after_hook_tl` will be set during the composition of the box `\l_@@_cell_box` and will be used *after* the composition in order to modify that box.

```
1188   \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

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

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

```
1190   \int_gincr:N \c@jCol
```

Now, we increment the counter of the rows. We don't do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don't want to take into account.

```

1191   \int_compare:nNnT \c@jCol = 1
1192     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }

```

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

```

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

```

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

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

```

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

```

The following macro `\@@_begin_of_row` is usually used in the cell number 1 of the row. However, when the key `first-col` is used, `\@@_begin_of_row` is executed in the cell number 0 of the row.

```

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

```

Remark: If the key `recreate-cell-nodes` of the `\CodeBefore` is used, then we will add some lines to that command.

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command.

```

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

```

```

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

```

```

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

```

The token list `\g_@@_cell_after_hook_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```

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

```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

```

1314 \@@_update_max_cell_width:

```

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

```

1315 \@@_update_for_first_and_last_row:

```

If the cell is empty, or may be considered as if, we must not create the PGF node, for two reasons:

- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

However, it’s very difficult to determine whether a cell is empty. Up to now we use the following technic:

- for the columns of type p, m, b, V (of `varwidth`) or X, we test whether the cell is syntactically empty with `\@@_test_if_empty:` and `\@@_test_if_empty_for_S:`
- if the width of the box `\l_@@_cell_box` (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a `\rlap`, `\llap`, `\clap` or a `\mathclap` of `mathtools`).
- the cells with a command `\Ldots` or `\Cdots`, `\Vdots`, etc., should also be considered as empty; if `nullify-dots` is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of `\CodeAfter`); however, if `nullify-dots` is not in force, a phantom of `\ldots`, `\cdots`, `\vdots` is inserted and its width is not equal to zero; that’s why these commands raise a boolean `\g_@@_empty_cell_bool` and we begin by testing this boolean.

```

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

```

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

```

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

```

The following variant of `\@@_cell_end:` is only for the columns of type `w{s}{...}` or `W{s}{...}` (which use the horizontal alignment key `s` of `\makebox`).

```

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

```

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

```

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

```

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

```

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

```


As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_the_cell:` when the construction of the cell nodes (of the form $(i-j)$) in the `\CodeBefore` is required.

```

1379 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1380 {
1381   \cs_new_protected:Npn \@@_patch_node_for_cell:
1382   {
1383     \hbox_set:Nn \l_@@_cell_box
1384     {
1385       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1386       \hbox_overlap_left:n
1387       {
1388         \pgfsys@markposition
1389         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

I don't know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way `latex`, `divps`, `ps2pdf` (or Adobe Distiller). However, it seems to work.

```

1390       #1
1391     }
1392     \box_use:N \l_@@_cell_box
1393     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1394     \hbox_overlap_left:n
1395     {
1396       \pgfsys@markposition
1397       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1398     }
1399     #1
1400   }
1401 }
1402 }

```

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

```

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

```

The second argument of the following command `\@@_instruction_of_type:nnn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The third argument is the list of options. This command writes in the corresponding `\g_@@_type_lines_tl` the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots[color=red] & & 
\end{pNiceMatrix}

```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & & 6 \\ 7 & \cdots & & \end{pmatrix}$$

the content of `\g_@@_Cdots_lines_tl` will be:

```

\@@_draw_Cdots:nnn {2}{2}{}
\@@_draw_Cdots:nnn {3}{2}{color=red}

```

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions (with consequences for the parallelisation of the diagonal lines).

```

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

```

```

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

```

```

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

```

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

```

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

```

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

```

1428 \@tabarray

```

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

```

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

```

We keep in memory the standard version of `\ialign` because we will redefine `\ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array.

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

We don't draw now the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for theses rules (the rules will be drawn by PGF).

```

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

```

The counter `\c@iRow` has the value `-1` only if there is a “first row” and that we are before that “first row”, i.e. just before the beginning of the array.

```

1485     \int_compare:nNnT \c@iRow > { -1 }
1486     {
1487         \int_compare:nNnF \c@iRow = \l_@@_last_row_int

```

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

```

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

```

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

```

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

```

```

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

```

We will issue an error only during the first run.

```

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

```

```

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

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁴.

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

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

```

The environment `{array}` uses internally the command `\ialign`. We change the definition of `\ialign` for several reasons. In particular, `\ialign` sets `\everycr` to `{ }` and we *need* to have to change the value of `\everycr`.

```

1558   \cs_set_nopar:Npn \ialign
1559   {
1560       \@@_redefine_everycr:
1561       \tabskip = \c_zero_skip

```

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

```

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

```

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

```

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

```

We keep in memory the old versions of `\ldots`, `\cdots`, etc. only because we use them inside `\phantom` commands in order that the new commands `\Ldots`, `\Cdots`, etc. give the same spacing (except when the option `nullify-dots` is used).

```

1576   \cs_set_eq:NN \@@_old_ldots \ldots
1577   \cs_set_eq:NN \@@_old_cdots \cdots
1578   \cs_set_eq:NN \@@_old_vdots \vdots
1579   \cs_set_eq:NN \@@_old_ddots \ddots

```

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

```

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

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition.

```

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

```

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

```

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

```

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

```

1619 \seq_gclear:N \g_@@_multicolumn_cells_seq
1620 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

The counter `\c@iRow` will be used to count the rows of the array (its incrementation will be in the first cell of the row).

```

1621 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

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

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

```

1622 \int_gzero_new:N \g_@@_row_total_int

```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_cell_begin:w` executed at the beginning of each cell.

```

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

```

During the construction of the array, the instructions `\Cdots`, `\Ldots`, etc. will be written in token lists `\g_@@_Cdots_lines_tl`, etc. which will be executed after the construction of the array.

```

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

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

```

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

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

```

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

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-column` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it's possible only after the first compilation).

```

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

```

If there is an exterior row, we patch a command used in `\@@_cell_begin:w` in order to keep track of some dimensions needed to the construction of that “last row”.

```

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

```

```

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

```

Now the `\CodeBefore`.

```

1664 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

The value of `\g_@@_pos_of_blocks_seq` has been written on the aux file and loaded before the (potential) execution of the `\CodeBefore`. Now, we clear that variable because it will be reconstructed during the creation of the array.

```

1665 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

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

```

The command `\create_row_node:` will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avoid warnings or, maybe, errors). That’s why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```

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

```

The value `−2` is important.

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

```

1669 \@@_pre_array_ii:

```

The array will be composed in a box (named `\l_@@_the_array_box`) because we have to do manipulations concerning the potential exterior rows.

```

1670 \box_clear_new:N \l_@@_the_array_box

```

We compute the width of both delimiters. We remind that, when the environment `{NiceArray}` is used, it’s possible to specify the delimiters in the preamble (eg `[ccc]`).

```

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

```

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

```

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

```

Here is the beginning of the box which will contain the array. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the second part of the environment (and the closing `\c_math_toggle_token` also).

```

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

```


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

```
1693 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1694 {
1695     \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1696     \bool_set_true:N \l_@@_code_before_bool
```

We go on with `\@@_pre_array:` which will (among other) execute the `\CodeBefore` (specified in the key `code-before` or after the keyword `\CodeBefore`). By definition, the `\CodeBefore` must be executed before the body of the array...

```
1697     \@@_pre_array:
1698 }
```

10 The `\CodeBefore`

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

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

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```
1701     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1702     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1703     \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1704     \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
```

Now, we will create all the `col` nodes and `row` nodes with the informations written in the `aux` file. You use the technique described in the page 1229 of `pgfmanual.pdf`, version 3.1.4b.

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

First, the recreation of the `row` nodes.

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

Now, the recreation of the `col` nodes.

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

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

```
1721     \@@_create_diag_nodes:
```

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

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

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

```

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

```

```

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

```

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

```

1752   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token

```

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

```

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

```

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

```

1758   \exp_last_unbraced:NV \@@_CodeBefore_keys:
1759   \g_@@_pre_code_before_tl

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

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

```

```

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

```

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

```

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

```

By default, if the user uses the `\CodeBefore`, only the `col` nodes, `row` nodes and `diag` nodes are available in that `\CodeBefore`. With the key `create-cell-nodes`, the cell nodes, that is to say the nodes of the form `(i-j)` (but not the extra nodes) are also available because those nodes also are recreated and that recreation is done by the following command.

```

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

```

```

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

```

```

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

```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (#5) which is the name of the block, is not empty.⁶

```

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

```

```

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

```

⁶Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

11 The environment `{NiceArrayWithDelims}`

```

1875 \NewDocumentEnvironment { NiceArrayWithDelims }
1876 { m m O { } m ! O { } t \CodeBefore }
1877 {
1878   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1879   \@@_provide_pgfsyspdfmark:
1880   \bool_if:NT \g_@@_footnote_bool \savenotes

```

The aim of the following `\bgroup` (the corresponding `\egroup` is, of course, at the end of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

1881   \bgroup

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

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

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁷. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```

1896   \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

```

We deactivate Tikz externalization because we will use PGF pictures with the options `overlay` and `remember picture` (or equivalent forms). We deactivate with `\tikzexternaldisable` and not with `\tikzset{external/export=false}` which is *not* equivalent.

```

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

```

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

```

1903   \int_gincr:N \g_@@_env_int
1904   \bool_if:NF \l_@@_block_auto_columns_width_bool
1905   { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

The sequence `\g_@@_blocks_seq` will contain the carateristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks (except the blocks with the key `hvlines`).

```

1906   \seq_gclear:N \g_@@_blocks_seq
1907   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

In fact, the sequence `\g_@@_pos_of_blocks_seq` will also contain the positions of the cells with a `\diagbox`.

```

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

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1910 \tl_gclear_new:N \g_@@_code_before_tl
1911 \tl_gclear:N \g_@@_row_style_tl

```

We load all the informations written in the aux file during previous compilations corresponding to the current environment.

```

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

```

Now, we prepare the token list for the instructions that we will have to write on the aux file at the end of the environment.

```

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

```

The set of keys is not exactly the same for {NiceArray} and for the variants of {NiceArray} ({pNiceArray}, {bNiceArray}, etc.) because, for {NiceArray}, we have the options t, c, b and baseline.

```

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

```

```

1930 \@@_set_CT@arc@:V \l_@@_rules_color_tl

```

The argument #6 is the last argument of {NiceArrayWithDelims}. With that argument of type “t \CodeBefore”, we test whether there is the keyword \CodeBefore at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword \CodeBefore and the (other) keyword \Body. It's the job that will do the command \@@_CodeBefore_Body:w. After that job, the command \@@_CodeBefore_Body:w will go on with \@@_pre_array:.

```

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

```

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

```

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

```

End of the construction of the array (in the box \l_@@_the_array_box).

If the user has used the key width without any column X, we raise an error.

```

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

```

Now, if there is at least one X-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, \l_@@_X_columns_dim will be the width of a column of weight 1. For a X-column of weight n , the width will be \l_@@_X_columns_dim multiplied by n .

```

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

```

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

```

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

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` change: `\c@jCol` will be the number of columns without the “last column”; `\g_@@_col_total_int` will be the number of columns with this “last column”.⁸

```

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

```

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

```

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

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 87).

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

We compute `\l_tmpb_dim` which is the total height of the “last row” below the array (when the key `last-row` is used). A value of `-2` for `\l_@@_last_row_int` means that there is no “last row”.⁹

```

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

```

We take into account the “first row” (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here.

```

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

```

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

```

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

```

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

```

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

```

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


```

2041         \c_math_toggle_token
2042     }

```

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

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

```

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

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 88).

```

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

```

The aim of the following `\egroup` (the corresponding `\bgroup` is, of course, at the beginning of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

2061     \egroup

```

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

```

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

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

```

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

12 We construct the preamble of the array

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

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

```

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

```

```

2078 \cs_new_protected:Npn \@@_transform_preamble_i:
2079 {
2080   \int_gzero:N \c@jCol

```

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

```

2081   \seq_gclear:N \g_@@_cols_vlism_seq

```

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

```

2082   \bool_gset_false:N \g_tmpb_bool

```

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

```

2083   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

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

```

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

```

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

2100   \@@_replace_columncolor:
2101 }

```

```

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

```

```

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

```

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

```

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

```

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

```

2126 \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }

```

We complete the preamble with the potential “exterior columns” (on both sides).

```

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

```

We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in `{NiceTabular*}` (we control that with the value of `\l_@@_tabular_width_dim`).

```

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

```

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

```

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

```

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

```

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

```

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

Now, the columns defined by \newcolumntype of array.

```

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

```

For c, l and r

```

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

```

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

```

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

```

For ! and @

```

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

```

For |

```

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

```

\l_tmpa_int is the number of successive occurrences of |

```

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

```

```

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

```

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

```

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

```

We don't have provided value for `start` nor for `end`, which means that the rule will cover (potentially) all the rows of the array.

```

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

```

```

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

```

The specifier `p` (and also the specifiers `m`, `b`, `V` and `X`) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys.

```

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

```

```

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

```

For p, b and m.

```

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

```

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

```

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

```

#1 is the optional argument of the specifier (a list of *key-value* pairs).

#2 is the mandatory argument of the specifier: the width of the column.

```

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

```

The possible values of \l_@@_hpos_col_str are j (for *justified* which is the initial value), l, c, r, L, C and R (when the user has used the corresponding key in the optional argument of the specifier).

```

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

```

The first argument is the width of the column. The second is the type of environment: minipage or varwidth.

```

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

```

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

```

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

```

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

```

2342      { \str_lowercase:V \l_@@_hpos_col_str }
2343  }
2344  }

```

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

```

2345      \int_gincr:N \c@jCol
2346      \@@_rec_preamble_after_col:n
2347  }

```

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

```

2348 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2349 {
2350     \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2351     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2352     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2353     \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2354     \tl_gclear:N \g_@@_pre_cell_tl
2355     \tl_gput_right:Nn \g_@@_array_preamble_tl
2356     {

```

2357 > {

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.

```
2358 \dim_set:Nn \l_@@_col_width_dim { #2 }
2359 \@@_cell_begin:w
2360 \begin { #7 } [ #1 ] { #2 }
```

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

```
2361 \everypar
2362 {
2363 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2364 \everypar { }
2365 }
```

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

2366 #3

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

```
2367 \g_@@_row_style_tl
2368 \arraybackslash
2369 #5
2370 }
2371 #8
2372 < {
2373 #6
```

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

```
2374 \@finalstrut \@arstrutbox
2375 % \bool_if:NT \g_@@_rotate_bool { \raggedright \hspace = 3 cm }
2376 \end { #7 }
```

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

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

```
2382 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2383 {
2384 \peek_meaning:NT \unskip
2385 {
2386 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2387 {
2388 \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
```

We put the following code in order to have a column with the correct width even when all the cells of the column are empty.

```
2389 \skip_horizontal:N \l_@@_col_width_dim
2390 }
2391 }
2392 #1
2393 }

2394 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2395 {
2396 \peek_meaning:NT \__siunitx_table_skip:n
2397 {
2398 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2399 { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2400 }
2401 #1
2402 }
```


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.

```
2403 \cs_new_protected:Npn \@_center_cell_box:
2404 {
```

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

```
2405 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2406 {
2407   \int_compare:nNnT
2408     { \box_ht:N \l_@@_cell_box }
2409     >
```

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).

```
2410   { \box_ht:N \strutbox }
2411   {
2412     \hbox_set:Nn \l_@@_cell_box
2413       {
2414         \box_move_down:nn
2415           {
2416             ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2417               + \baselineskip ) / 2
2418           }
2419         { \box_use:N \l_@@_cell_box }
2420       }
2421     }
2422   }
2423 }
```

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

```
2424 \cs_new:Npn \@_V #1 #2
2425 {
2426   \str_if_eq:nnTF { #2 } { [ ] }
2427     { \@_make_preamble_V_i:w [ ] }
2428     { \@_make_preamble_V_i:w [ ] { #2 } }
2429 }
2430 \cs_new_protected:Npn \@_make_preamble_V_i:w [ #1 ]
2431 { \@_make_preamble_V_ii:nn { #1 } }
2432 \cs_new_protected:Npn \@_make_preamble_V_ii:nn #1 #2
2433 {
2434   \str_set:Nn \l_@@_vpos_col_str { p }
2435   \str_set:Nn \l_@@_hpos_col_str { j }
2436   \tl_set:Nn \l_tmpa_tl { #1 }
2437   \@_keys_p_column:V \l_tmpa_tl
2438   \IfPackageLoadedTF { varwidth }
2439     { \@_make_preamble_ii_iv:nn { #2 } { varwidth } }
2440     {
2441       \@_error_or_warning:n { varwidth-not-loaded }
2442       \@_make_preamble_ii_iv:nn { #2 } { minipage }
2443     }
2444 }
```

For `w` and `W`

```
2445 \cs_new:Npn \@_w { \@_make_preamble_w:nnnn { } }
2446 \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.

```

2447 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2448 {
2449   \str_if_eq:nnTF { #3 } { s }
2450   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2451   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2452 }

```

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.

```

2453 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2454 {
2455   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2456   \tl_gclear:N \g_@@_pre_cell_tl
2457   \tl_gput_right:Nn \g_@@_array_preamble_tl
2458   {
2459     > {
2460       \dim_set:Nn \l_@@_col_width_dim { #2 }
2461       \@@_cell_begin:w
2462       \str_set:Nn \l_@@_hpos_cell_str { c }
2463     }
2464     c
2465     < {
2466       \@@_cell_end_for_w_s:
2467       #1
2468       \@@_adjust_size_box:
2469       \box_use_drop:N \l_@@_cell_box
2470     }
2471   }
2472   \int_gincr:N \c@jCol
2473   \@@_rec_preamble_after_col:n
2474 }

```

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

```

2475 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2476 {
2477   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2478   \tl_gclear:N \g_@@_pre_cell_tl
2479   \tl_gput_right:Nn \g_@@_array_preamble_tl
2480   {
2481     > {

```

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.

```

2482       \dim_set:Nn \l_@@_col_width_dim { #4 }
2483       \hbox_set:Nw \l_@@_cell_box
2484       \@@_cell_begin:w
2485       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2486     }
2487     c
2488     < {
2489       \@@_cell_end:
2490       \hbox_set_end:
2491       #1
2492       \@@_adjust_size_box:
2493       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2494     }
2495   }

```

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

```

2496 \int_gincr:N \c@jCol
2497 \@@_rec_preamble_after_col:n
2498 }

2499 \cs_new_protected:Npn \@@_special_W:
2500 {
2501   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2502   { \@@_warning:n { W~warning } }
2503 }

```

For S (of siunitx).

```

2504 \cs_new:Npn \@@_S #1 #2
2505 {
2506   \str_if_eq:nnTF { #2 } { [ ] }
2507   { \@@_make_preamble_S:w [ ] }
2508   { \@@_make_preamble_S:w [ ] { #2 } }
2509 }

2510 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2511 { \@@_make_preamble_S_i:n { #1 } }

2512 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2513 {
2514   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2515   \tl_gclear:N \g_@@_pre_cell_tl
2516   \tl_gput_right:Nn \g_@@_array_preamble_tl
2517   {
2518     > {
2519       \@@_cell_begin:w
2520       \keys_set:nn { siunitx } { #1 }
2521       \siunitx_cell_begin:w
2522     }
2523     c
2524     < { \siunitx_cell_end: \@@_cell_end: }
2525   }

```

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

```

2526 \int_gincr:N \c@jCol
2527 \@@_rec_preamble_after_col:n
2528 }

```

For (, [and \{.

```

2529 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2530 {
2531   \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.

```

2532 \int_if_zero:nTF \c@jCol
2533 {
2534   \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2535   {

```

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

```

2536   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2537   \tl_gset:Nn \g_@@_right_delim_tl { . }
2538   \@@_rec_preamble:n #2
2539 }
2540 {
2541   \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2542   \@@_make_preamble_iv:nn { #1 } { #2 }
2543 }
2544 }

```

```

2545     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2546   }
2547   \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2548   \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2549   \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2550   {
2551     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2552     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2553     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2554     {
2555       \@@_error:nn { delimiter-after-opening } { #2 }
2556       \@@_rec_preamble:n
2557     }
2558     { \@@_rec_preamble:n #2 }
2559   }

```

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

```

2560   \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}`).

```

2561   \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2562   {
2563     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter-with-small } }
2564     \tl_if_in:nnTF { ) ] \} } { #2 }
2565     { \@@_make_preamble_v:nnn #1 #2 }
2566     {
2567       \tl_if_eq:nnTF { \stop } { #2 }
2568       {
2569         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2570         { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2571         {
2572           \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2573           \tl_gput_right:Nx \g_@@_pre_code_after_tl
2574           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2575           \@@_rec_preamble:n #2
2576         }
2577       }
2578       {
2579         \tl_if_in:nnT { ( [ \{ \left } { #2 }
2580         { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2581         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2582         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2583         \@@_rec_preamble:n #2
2584       }
2585     }
2586   }
2587   \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ] }
2588   \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2589   \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2590   {
2591     \tl_if_eq:nnTF { \stop } { #3 }
2592     {
2593       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2594       {
2595         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2596         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2597         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2598         \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2599       }

```

```

2600     {
2601         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2602         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2603             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2604         \@@_error:nn { double~closing~delimiter } { #2 }
2605     }
2606 }
2607 {
2608     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2609         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2610     \@@_error:nn { double~closing~delimiter } { #2 }
2611     \@@_rec_preamble:n #3
2612 }
2613 }

2614 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2615     { \use:c { @@ _ \token_to_str:N ) } }

2616 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2617     {
2618         \str_if_eq:nnTF { #1 } { < }
2619             \@@_rec_preamble_after_col_i:n
2620         {
2621             \str_if_eq:nnTF { #1 } { @ }
2622                 \@@_rec_preamble_after_col_ii:n
2623             {
2624                 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2625                     {
2626                         \tl_gput_right:Nn \g_@@_array_preamble_tl
2627                             { ! { \skip_horizontal:N \arrayrulewidth } }
2628                     }
2629                     {
2630                         \exp_args:NNe
2631                         \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2632                             {
2633                                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2634                                     { ! { \skip_horizontal:N \arrayrulewidth } }
2635                             }
2636                     }
2637                 \@@_rec_preamble:n { #1 }
2638             }
2639         }
2640     }

2641 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2642     {
2643         \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2644         \@@_rec_preamble_after_col:n
2645     }

```

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.

```

2646 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2647     {
2648         \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2649             {
2650                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2651                     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2652             }
2653             {

```

```

2654 \exp_args:NNe
2655 \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2656 {
2657     \tl_gput_right:Nn \g_@@_array_preamble_tl
2658     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2659 }
2660 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2661 }
2662 \@@_rec_preamble:n
2663 }

2664 \cs_new:cpn { @@ _ * } #1 #2 #3
2665 {
2666     \tl_clear:N \l_tmpa_tl
2667     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2668     \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpa_tl
2669 }

```

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

```

2670 \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`.

```

2671 \cs_new:Npn \@@_X #1 #2
2672 {
2673     \str_if_eq:nnTF { #2 } { [ ]
2674         { \@@_make_preamble_X:w [ ]
2675           { \@@_make_preamble_X:w [ ] #2 }
2676         }
2677     \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2678     { \@@_make_preamble_X_i:n { #1 } }

```

`#1` is the optional argument of the `X` specifier (a list of *key-value* pairs).

The following set of keys is for the specifier `X` in the preamble of the array. Such specifier may have as keys all the keys of `{ WithArrows / p-column }` but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter `\l_@@_weight_int`).

```

2679 \keys_define:nn { WithArrows / X-column }
2680 { 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`.

```

2681 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2682 {

```

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`).

```

2683     \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`).

```

2684     \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`.

```

2685     \int_zero_new:N \l_@@_weight_int
2686     \int_set:Nn \l_@@_weight_int { 1 }
2687     \tl_set:Nn \l_tmpa_tl { #1 }
2688     \@@_keys_p_column:V \l_tmpa_tl
2689     \keys_set:nV { WithArrows / X-column } \l_tmpa_tl

```

```

2690 \int_compare:nNnT \l_@@_weight_int < 0
2691 {
2692   \@@_error_or_warning:n { negative-weight }
2693   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2694 }
2695 \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).

```

2696 \bool_if:NTF \l_@@_X_columns_aux_bool
2697 {
2698   \exp_args:Nne
2699   \@@_make_preamble_ii_iv:nn
2700   { \l_@@_weight_int \l_@@_X_columns_dim }
2701   { minipage }
2702 }
2703 {
2704   \tl_gput_right:Nn \g_@@_array_preamble_tl
2705   {
2706     > {
2707       \@@_cell_begin:w
2708       \bool_set_true:N \l_@@_X_column_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`.

```

2709 \NotEmpty

```

The following code will nullify the box of the cell.

```

2710 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2711 { \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`.

```

2712 \begin { minipage } { 5 cm } \arraybackslash
2713 }
2714 c
2715 < {
2716   \end { minipage }
2717   \@@_cell_end:
2718 }
2719 }
2720 \int_gincr:N \c@jCol
2721 \@@_rec_preamble_after_col:n
2722 }
2723 }

```

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

```

2724 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2725 {
2726   \seq_gput_right:Nx \g_@@_cols_vlism_seq
2727   { \int_eval:n { \c@jCol + 1 } }
2728   \tl_gput_right:Nx \g_@@_array_preamble_tl
2729   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2730   \@@_rec_preamble:n
2731 }

```

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.

```

2732 \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).

```

2733 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2734 { \@@_fatal:n { Preamble~forgotten } }
2735 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2736 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2737 \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).

```

2738 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2739 {

```

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`.

```

2740 \multispan { #1 }
2741 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2742 \begingroup
2743 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

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

```

2744 \tl_gclear:N \g_@@_preamble_tl
2745 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2746 \exp_args:NV \@mkpream \g_@@_preamble_tl
2747 \@addtopreamble \@empty
2748 \endgroup

```

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

```

2749 \int_compare:nNnT { #1 } > 1
2750 {
2751   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2752   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2753   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2754   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2755   {
2756     {
2757       \int_if_zero:nTF \c@jCol
2758       { \int_eval:n { \c@iRow + 1 } }
2759       { \int_use:N \c@iRow }
2760     }
2761     { \int_eval:n { \c@jCol + 1 } }
2762     {
2763       \int_if_zero:nTF \c@jCol
2764       { \int_eval:n { \c@iRow + 1 } }
2765       { \int_use:N \c@iRow }
2766     }
2767     { \int_eval:n { \c@jCol + #1 } }
2768     { } % for the name of the block
2769   }
2770 }

```


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

```
2771 \cs_set:Npn \@sharp { #3 }
2772 \@arstrut
2773 \@preamble
2774 \null
```

We add some lines.

```
2775 \int_gadd:Nn \c@jCol { #1 - 1 }
2776 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2777 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2778 \ignorespaces
2779 }
```

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`.

```
2780 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2781 {
2782   \str_case:nnF { #1 }
2783   {
2784     c { @@_make_m_preamble_i:n #1 }
2785     l { @@_make_m_preamble_i:n #1 }
2786     r { @@_make_m_preamble_i:n #1 }
2787     > { @@_make_m_preamble_ii:nn #1 }
2788     ! { @@_make_m_preamble_ii:nn #1 }
2789     @ { @@_make_m_preamble_ii:nn #1 }
2790     | { @@_make_m_preamble_iii:n #1 }
2791     p { @@_make_m_preamble_iv:nnn t #1 }
2792     m { @@_make_m_preamble_iv:nnn c #1 }
2793     b { @@_make_m_preamble_iv:nnn b #1 }
2794     w { @@_make_m_preamble_v:nnnn { } #1 }
2795     W { @@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2796     \q_stop { }
2797   }
2798   {
2799     \cs_if_exist:cTF { NC @ find @ #1 }
2800     {
2801       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2802       \exp_last_unbraced:NV \@@_make_m_preamble:n \l_tmpa_tl
2803     }
2804     {
2805       \tl_if_eq:nnT { #1 } { S }
2806       { @@_fatal:n { unknown~column~type~S } }
2807       { @@_fatal:nn { unknown~column~type } { #1 } }
2808     }
2809   }
2810 }
```

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

```
2811 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2812 {
2813   \tl_gput_right:Nn \g_@@_preamble_tl
2814   {
2815     > { @@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2816     #1
2817     < @@_cell_end:
2818   }
```

We test for the presence of a `<`.

```
2819   @@_make_m_preamble_x:n
2820 }
```

For >, ! and @

```

2821 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2822 {
2823   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2824   \@@_make_m_preamble:n
2825 }

```

For |

```

2826 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2827 {
2828   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2829   \@@_make_m_preamble:n
2830 }

```

For p, m and b

```

2831 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2832 {
2833   \tl_gput_right:Nn \g_@@_preamble_tl
2834   {
2835     > {
2836       \@@_cell_begin:w
2837       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2838       \mode_leave_vertical:
2839       \arraybackslash
2840       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2841     }
2842     c
2843     < {
2844       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2845       \end { minipage }
2846       \@@_cell_end:
2847     }
2848   }

```

We test for the presence of a <.

```

2849   \@@_make_m_preamble_x:n
2850 }

```

For w and W

```

2851 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2852 {
2853   \tl_gput_right:Nn \g_@@_preamble_tl
2854   {
2855     > {
2856       \dim_set:Nn \l_@@_col_width_dim { #4 }
2857       \hbox_set:Nw \l_@@_cell_box
2858       \@@_cell_begin:w
2859       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2860     }
2861     c
2862     < {
2863       \@@_cell_end:
2864       \hbox_set_end:
2865       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2866       #1
2867       \@@_adjust_size_box:
2868       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2869     }
2870   }

```

We test for the presence of a <.

```

2871   \@@_make_m_preamble_x:n
2872 }

```

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

```

2873 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2874 {
2875   \str_if_eq:nnTF { #1 } { < }
2876     \@@_make_m_preamble_ix:n
2877     { \@@_make_m_preamble:n { #1 } }
2878 }
2879 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2880 {
2881   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2882   \@@_make_m_preamble_x:n
2883 }

```

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`).

```

2884 \cs_new_protected:Npn \@@_put_box_in_flow:
2885 {
2886   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2887   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2888   \tl_if_eq:nnTF \l_@@_baseline_tl { c }
2889     { \box_use_drop:N \l_tmpa_box }
2890     \@@_put_box_in_flow_i:
2891 }

```

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).

```

2892 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2893 {
2894   \pgfpicture
2895     \@@_qpoint:n { row - 1 }
2896     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2897     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2898     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2899     \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).

```

2900   \str_if_in:nnTF \l_@@_baseline_tl { line- }
2901   {
2902     \int_set:Nn \l_tmpa_int
2903     {
2904       \str_range:Nnn
2905         \l_@@_baseline_tl
2906         6
2907         { \tl_count:V \l_@@_baseline_tl }
2908     }
2909     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2910   }
2911   {
2912     \str_case:VnF \l_@@_baseline_tl
2913     {
2914       { t } { \int_set:Nn \l_tmpa_int 1 }
2915       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2916     }
2917     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2918     \bool_lazy_or:nnT
2919     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2920     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2921     {
2922       \@@_error:n { bad~value~for~baseline }

```

```

2923         \int_set:Nn \l_tmpa_int 1
2924     }
2925     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2926         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2927     }
2928     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2929     \endpgfpicture
2930     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2931     \box_use_drop:N \l_tmpa_box
2932 }

```

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).

```

2933 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2934 {

```

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.

```

2935     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2936     {
2937         \int_compare:nNnT \c@jCol > 1 % added 2023-08-13
2938         {
2939             \box_set_wd:Nn \l_@@_the_array_box
2940             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2941         }
2942     }

```

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

```

2943     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2944     \bool_if:NT \l_@@_caption_above_bool
2945     {
2946         \tl_if_empty:NF \l_@@_caption_tl
2947         {
2948             \bool_set_false:N \g_@@_caption_finished_bool
2949             \int_gzero:N \c@tabularnote
2950             \@@_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).

```

2951         \int_compare:nNnT \g_@@_notes_caption_int > 0
2952         {
2953             \tl_build_gput_right:Nx \g_@@_aux_tl
2954             {
2955                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2956                 { \int_use:N \g_@@_notes_caption_int }
2957             }
2958             \int_gzero:N \g_@@_notes_caption_int
2959         }
2960     }
2961 }

```

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

```

2962     \hbox
2963     {
2964         \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.

```

2965     \@@_create_extra_nodes:
2966     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2967 }

```

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).

```

2968 \bool_lazy_any:nT
2969 {
2970   { ! \seq_if_empty_p:N \g_@@_notes_seq }
2971   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2972   { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2973 }
2974 \@@_insert_tabularnotes:
2975 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2976 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2977 \end { minipage }
2978 }

```

```

2979 \cs_new_protected:Npn \@@_insert_caption:
2980 {
2981   \tl_if_empty:NF \l_@@_caption_tl
2982   {
2983     \cs_if_exist:NTF \@capytype
2984     { \@@_insert_caption_i: }
2985     { \@@_error:n { caption-outside~float } }
2986   }
2987 }

```

```

2988 \cs_new_protected:Npn \@@_insert_caption_i:
2989 {
2990   \group_begin:

```

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

```

2991 \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.

```

2992 \IfPackageLoadedTF { floatrow }
2993 { \cs_set_eq:NN \@makecaption \FR@makecaption }
2994 { }
2995 \tl_if_empty:NTF \l_@@_short_caption_tl
2996 { \caption }
2997 { \caption [ \l_@@_short_caption_tl ] }
2998 { \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.

```

2999 \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
3000 {
3001   \bool_gset_true:N \g_@@_caption_finished_bool
3002   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3003   \int_gzero:N \c@tabularnote

```

```

3004     }
3005     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3006     \group_end:
3007 }

3008 \cs_new_protected:Npn \@@_tabulernote_error:n #1
3009 {
3010     \@@_error_or_warning:n { tabulernote~below~the~tabular }
3011     \@@_gredirect_none:n { tabulernote~below~the~tabular }
3012 }

3013 \cs_new_protected:Npn \@@_insert_tabularnotes:
3014 {
3015     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3016     \int_set:Nn \c@tabulernote { \seq_count:N \g_@@_notes_seq }
3017     \skip_vertical:N 0.65ex

```

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

```

3018     \group_begin:
3019     \l_@@_notes_code_before_tl
3020     \tl_if_empty:NF \g_@@_tabulernote_tl
3021     {
3022         \g_@@_tabulernote_tl \par
3023         \tl_gclear:N \g_@@_tabulernote_tl
3024     }

```

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.

```

3025     \int_compare:nNnT \c@tabulernote > 0
3026     {
3027         \bool_if:NTF \l_@@_notes_para_bool
3028         {
3029             \begin { tabularnotes* }
3030             \seq_map_inline:Nn \g_@@_notes_seq
3031             { \@@_one_tabulernote:nn ##1 }
3032             \strut
3033             \end { tabularnotes* }

```

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

```

3034         \par
3035     }
3036     {
3037         \tabularnotes
3038         \seq_map_inline:Nn \g_@@_notes_seq
3039         { \@@_one_tabulernote:nn ##1 }
3040         \strut
3041         \endtabularnotes
3042     }
3043 }
3044 \unskip
3045 \group_end:
3046 \bool_if:NT \l_@@_notes_bottomrule_bool
3047 {
3048     \IfPackageLoadedTF { booktabs }
3049     {

```

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

```

3050         \skip_vertical:N \aboverulesep

```

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

```

3051         { \CT@arc@ \hrule height \heavyrulewidth }
3052     }
3053     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3054 }
3055 \l_@@_notes_code_after_tl

```

```

3056 \seq_gclear:N \g_@@_notes_seq
3057 \seq_gclear:N \g_@@_notes_in_caption_seq
3058 \int_gzero:N \c@tabularnote
3059 }

```

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

```

3060 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3061 {
3062   \tl_if_novalue:nTF { #1 }
3063     { \item }
3064     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3065 }

```

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`.

```

3066 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3067 {
3068   \pgfpicture
3069     \@@_qpoint:n { row - 1 }
3070     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3071     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3072     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3073   \endpgfpicture
3074   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3075   \int_if_zero:nT \l_@@_first_row_int
3076   {
3077     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3078     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3079   }
3080   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3081 }

```

Now, the general case.

```

3082 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3083 {

```

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

```

3084   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3085     { \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`.

```

3086   \pgfpicture
3087     \@@_qpoint:n { row - 1 }
3088     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3089     \str_if_in:NnTF \l_@@_baseline_tl { line- }
3090     {
3091       \int_set:Nn \l_tmpa_int
3092       {
3093         \str_range:Nnn
3094           \l_@@_baseline_tl
3095           6
3096           { \tl_count:V \l_@@_baseline_tl }
3097       }
3098       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3099     }
3100     {
3101       \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3102       \bool_lazy_or:nnT

```

```

3103     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3104     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3105     {
3106         \@@_error:n { bad~value~for~baseline }
3107         \int_set:Nn \l_tmpa_int 1
3108     }
3109     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3110 }
3111 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3112 \endpgfpicture
3113 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3114 \int_if_zero:nT \l_@@_first_row_int
3115 {
3116     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3117     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3118 }
3119 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3120 }

```

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.

```

3121 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3122 {

```

We will compute the real width of both delimiters used.

```

3123     \dim_zero_new:N \l_@@_real_left_delim_dim
3124     \dim_zero_new:N \l_@@_real_right_delim_dim
3125     \hbox_set:Nn \l_tmpb_box
3126     {
3127         \c_math_toggle_token
3128         \left #1
3129         \vcenter
3130         {
3131             \vbox_to_ht:nn
3132             { \box_ht_plus_dp:N \l_tmpa_box }
3133             { }
3134         }
3135         \right .
3136         \c_math_toggle_token
3137     }
3138     \dim_set:Nn \l_@@_real_left_delim_dim
3139     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3140     \hbox_set:Nn \l_tmpb_box
3141     {
3142         \c_math_toggle_token
3143         \left .
3144         \vbox_to_ht:nn
3145         { \box_ht_plus_dp:N \l_tmpa_box }
3146         { }
3147         \right #2
3148         \c_math_toggle_token
3149     }
3150     \dim_set:Nn \l_@@_real_right_delim_dim
3151     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3152     \skip_horizontal:N \l_@@_left_delim_dim
3153     \skip_horizontal:N -\l_@@_real_left_delim_dim
3154     \@@_put_box_in_flow:
3155     \skip_horizontal:N \l_@@_right_delim_dim
3156     \skip_horizontal:N -\l_@@_real_right_delim_dim
3157 }

```


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).

```
3158 \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).

```
3159 {
3160   \peek_remove_spaces:n
3161   {
3162     \peek_meaning:NTF \end
3163     \@@_analyze_end:Nn
3164     {
3165       \@@_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`).

```
3166       \exp_args:NV \@@_array: \g_@@_array_preamble_tl
3167     }
3168   }
3169 }
3170 {
3171   \@@_create_col_nodes:
3172   \endarray
3173 }
```

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

```
3174 \NewDocumentEnvironment { @@-light-syntax } { b }
3175 {
```

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`.

```
3176   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3177   \tl_map_inline:nn { #1 }
3178   {
3179     \str_if_eq:nnT { ##1 } { & }
3180     { \@@_fatal:n { ampersand-in-light-syntax } }
3181     \str_if_eq:nnT { ##1 } { \ }
3182     { \@@_fatal:n { double-backslash-in-light-syntax } }
3183   }
```

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`.

```
3184   \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3185 }
```

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.

```
3186 {
3187   \@@_create_col_nodes:
3188   \endarray
3189 }
```

```

3190 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3191 {
3192   \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).

```

3193   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3194   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3195   \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3196   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3197   \tl_if_empty:NF \l_tmpa_tl
3198   { \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.

```

3199   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3200   { \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`).

```

3201   \tl_build_begin:N \l_@@_new_body_tl
3202   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3203   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3204   \@@_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).

```

3205   \seq_map_inline:Nn \l_@@_rows_seq
3206   {
3207     \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3208     \@@_line_with_light_syntax:n { ##1 }
3209   }
3210   \tl_build_end:N \l_@@_new_body_tl
3211   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3212   {
3213     \int_set:Nn \l_@@_last_col_int
3214     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3215   }

```

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.

```

3216   \@@_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`).

```

3217   \exp_args:Nv \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3218   }
3219 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3220 {
3221   \seq_clear_new:N \l_@@_cells_seq
3222   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3223   \int_set:Nn \l_@@_nb_cols_int
3224   {
3225     \int_max:nn
3226     \l_@@_nb_cols_int
3227     { \seq_count:N \l_@@_cells_seq }
3228   }

```

```

3229 \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3230 \exp_args:NNV \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3231 \seq_map_inline:Nn \l_@@_cells_seq
3232 { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3233 }
3234 \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`.

```

3235 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3236 {
3237   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3238   { \@@_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.

```

3239   \end { #2 }
3240 }

```

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).

```

3241 \cs_new:Npn \@@_create_col_nodes:
3242 {
3243   \crrc
3244   \int_if_zero:nT \l_@@_first_col_int
3245   {
3246     \omit
3247     \hbox_overlap_left:n
3248     {
3249       \bool_if:NT \l_@@_code_before_bool
3250       { \pgfsys@markposition { \@@_env: - col - 0 } }
3251       \pgfpicture
3252       \pgfrememberpicturepositiononpagetrue
3253       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3254       \str_if_empty:NF \l_@@_name_str
3255       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3256       \endpgfpicture
3257       \skip_horizontal:N 2\col@sep
3258       \skip_horizontal:N \g_@@_width_first_col_dim
3259     }
3260     &
3261   }
3262   \omit

```

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

```

3263   \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`).

```

3264   \int_if_zero:nTF \l_@@_first_col_int
3265   {
3266     \bool_if:NT \l_@@_code_before_bool
3267     {
3268       \hbox
3269       {
3270         \skip_horizontal:N -0.5\arrayrulewidth
3271         \pgfsys@markposition { \@@_env: - col - 1 }
3272         \skip_horizontal:N 0.5\arrayrulewidth
3273       }
3274     }
3275     \pgfpicture

```

```

3276 \pgfrememberpicturerepositiononpagetrue
3277 \pgfcoordinate { \@@_env: - col - 1 }
3278 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3279 \str_if_empty:NF \l_@@_name_str
3280 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3281 \endpgfpicture
3282 }
3283 {
3284 \bool_if:NT \l_@@_code_before_bool
3285 {
3286 \hbox
3287 {
3288 \skip_horizontal:N 0.5\arrayrulewidth
3289 \pgfsys@markposition { \@@_env: - col - 1 }
3290 \skip_horizontal:N -0.5\arrayrulewidth
3291 }
3292 }
3293 \pgfpicture
3294 \pgfrememberpicturerepositiononpagetrue
3295 \pgfcoordinate { \@@_env: - col - 1 }
3296 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3297 \str_if_empty:NF \l_@@_name_str
3298 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3299 \endpgfpicture
3300 }

```

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 this variable in other cells (of the same row). The affectation of `\g_tmpa_skip`, like all the affectations, must be done after the `\omit` of the cell.

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but it will just after be erased by a fixed value in the concerned cases.

```

3301 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3302 \bool_if:NF \l_@@_auto_columns_width_bool
3303 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3304 {
3305 \bool_lazy_and:nnTF
3306 \l_@@_auto_columns_width_bool
3307 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3308 { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3309 { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3310 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3311 }
3312 \skip_horizontal:N \g_tmpa_skip
3313 \hbox
3314 {
3315 \bool_if:NT \l_@@_code_before_bool
3316 {
3317 \hbox
3318 {
3319 \skip_horizontal:N -0.5\arrayrulewidth
3320 \pgfsys@markposition { \@@_env: - col - 2 }
3321 \skip_horizontal:N 0.5\arrayrulewidth
3322 }
3323 }
3324 \pgfpicture
3325 \pgfrememberpicturerepositiononpagetrue
3326 \pgfcoordinate { \@@_env: - col - 2 }
3327 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3328 \str_if_empty:NF \l_@@_name_str
3329 { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3330 \endpgfpicture
3331 }

```

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.

```

3332 \int_gset:Nn \g_tmpa_int 1
3333 \bool_if:NTF \g_@@_last_col_found_bool
3334 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3335 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3336 {
3337   &
3338   \omit
3339   \int_gincr:N \g_tmpa_int

```

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

```

3340 \skip_horizontal:N \g_tmpa_skip
3341 \bool_if:NT \l_@@_code_before_bool
3342 {
3343   \hbox
3344   {
3345     \skip_horizontal:N -0.5\arrayrulewidth
3346     \pgfsys@markposition
3347     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3348     \skip_horizontal:N 0.5\arrayrulewidth
3349   }
3350 }

```

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

```

3351 \pgfpicture
3352 \pgfrememberpicturepositiononpagetrue
3353 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3354 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3355 \str_if_empty:NF \l_@@_name_str
3356 {
3357   \pgfnodealias
3358   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3359   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3360 }
3361 \endpgfpicture
3362 }

3363 &
3364 \omit

```

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

```

3365 \int_compare:nNnT \g_@@_col_total_int = 1
3366 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3367 \skip_horizontal:N \g_tmpa_skip
3368 \int_gincr:N \g_tmpa_int
3369 \bool_lazy_all:nT
3370 {
3371   { \bool_not_p:n \g_@@_delims_bool }
3372   { \bool_not_p:n \l_@@_tabular_bool }
3373   { \clist_if_empty_p:N \l_@@_vlines_clist }
3374   { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3375   { ! \l_@@_bar_at_end_of_pream_bool }
3376 }
3377 { \skip_horizontal:N -\col@sep }
3378 \bool_if:NT \l_@@_code_before_bool
3379 {
3380   \hbox
3381   {
3382     \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.

```

3383         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3384         { \skip_horizontal:N -\arraycolsep }
3385         \pgfsys@markposition
3386         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3387         \skip_horizontal:N 0.5\arrayrulewidth
3388         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3389         { \skip_horizontal:N \arraycolsep }
3390     }
3391 }
3392 \pgfpicture
3393 \pgfrememberpicturepositiononpagetrue
3394 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3395 {
3396     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3397     {
3398         \pgfpoint
3399         { - 0.5 \arrayrulewidth - \arraycolsep }
3400         \c_zero_dim
3401     }
3402     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3403 }
3404 \str_if_empty:NF \l_@@_name_str
3405 {
3406     \pgfnodealias
3407     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3408     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3409 }
3410 \endpgfpicture

3411 \bool_if:NT \g_@@_last_col_found_bool
3412 {
3413     \hbox_overlap_right:n
3414     {
3415         \skip_horizontal:N \g_@@_width_last_col_dim
3416         \bool_if:NT \l_@@_code_before_bool
3417         {
3418             \pgfsys@markposition
3419             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3420         }
3421         \pgfpicture
3422         \pgfrememberpicturepositiononpagetrue
3423         \pgfcoordinate
3424         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3425         \pgfpointorigin
3426         \str_if_empty:NF \l_@@_name_str
3427         {
3428             \pgfnodealias
3429             {
3430                 \l_@@_name_str - col
3431                 - \int_eval:n { \g_@@_col_total_int + 1 }
3432             }
3433             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3434         }
3435         \endpgfpicture
3436     }
3437 }
3438 \cr
3439 }
```

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

```

3440 \tl_const:Nn \c_@@_preamble_first_col_tl
3441 {
3442   >
3443   {

```

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).

```

3444     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3445     \bool_gset_true:N \g_@@_after_col_zero_bool
3446     \@@_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.

```

3447     \hbox_set:Nw \l_@@_cell_box
3448     \@@_math_toggle_token:
3449     \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”.

```

3450     \bool_lazy_and:nnT
3451     { \int_compare_p:nNn \c@iRow > 0 }
3452     {
3453       \bool_lazy_or_p:nn
3454       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3455       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3456     }
3457     {
3458       \l_@@_code_for_first_col_tl
3459       \xglobal \colorlet { nicematrix-first-col } { . }
3460     }
3461   }

```

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`.

```

3462   l
3463   <
3464   {
3465     \@@_math_toggle_token:
3466     \hbox_set_end:
3467     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3468     \@@_adjust_size_box:
3469     \@@_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.

```

3470     \dim_gset:Nn \g_@@_width_first_col_dim
3471     { \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.

```

3472     \hbox_overlap_left:n
3473     {
3474       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3475       \@@_node_for_cell:
3476       { \box_use_drop:N \l_@@_cell_box }
3477       \skip_horizontal:N \l_@@_left_delim_dim
3478       \skip_horizontal:N \l_@@_left_margin_dim
3479       \skip_horizontal:N \l_@@_extra_left_margin_dim
3480     }
3481     \bool_gset_false:N \g_@@_empty_cell_bool
3482     \skip_horizontal:N -2\col@sep
3483   }
3484 }

```

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

```

3485 \tl_const:Nn \c_@@_preamble_last_col_tl
3486 {
3487   >
3488   {
3489     \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).

```

3490     \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.

```

3491     \bool_gset_true:N \g_@@_last_col_found_bool
3492     \int_gincr:N \c@jCol
3493     \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.

```

3494     \hbox_set:Nw \l_@@_cell_box
3495     \@@_math_toggle_token:
3496     \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”.

```

3497     \int_compare:nNnT \c@iRow > 0
3498     {
3499       \bool_lazy_or:nnT
3500       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3501       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3502       {
3503         \l_@@_code_for_last_col_tl
3504         \xglobal \colorlet { nicematrix-last-col } { . }
3505       }
3506     }
3507   }
3508   1
3509   <
3510   {
3511     \@@_math_toggle_token:
3512     \hbox_set_end:
3513     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3514     \@@_adjust_size_box:
3515     \@@_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.

```

3516     \dim_gset:Nn \g_@@_width_last_col_dim
3517     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3518     \skip_horizontal:N -2\col@sep

```

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

```

3519     \hbox_overlap_right:n
3520     {
3521       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3522       {
3523         \skip_horizontal:N \l_@@_right_delim_dim
3524         \skip_horizontal:N \l_@@_right_margin_dim
3525         \skip_horizontal:N \l_@@_extra_right_margin_dim
3526         \@@_node_for_cell:
3527       }
3528     }
3529     \bool_gset_false:N \g_@@_empty_cell_bool
3530   }
3531 }

```


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

```

3532 \NewDocumentEnvironment { NiceArray } { }
3533 {
3534   \bool_gset_false:N \g_@@_delims_bool
3535   \str_if_empty:NT \g_@@_name_env_str
3536   { \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).

```

3537   \NiceArrayWithDelims . .
3538 }
3539 { \endNiceArrayWithDelims }

```

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

```

3540 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3541 {
3542   \NewDocumentEnvironment { #1 NiceArray } { }
3543   {
3544     \bool_gset_true:N \g_@@_delims_bool
3545     \str_if_empty:NT \g_@@_name_env_str
3546     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3547     \@@_test_if_math_mode:
3548     \NiceArrayWithDelims #2 #3
3549   }
3550   { \endNiceArrayWithDelims }
3551 }
3552 \@@_def_env:nnn p ( )
3553 \@@_def_env:nnn b [ ]
3554 \@@_def_env:nnn B \{ \}
3555 \@@_def_env:nnn v | |
3556 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3557 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3558 {
3559   \bool_set_false:N \l_@@_preamble_bool
3560   \tl_clear:N \l_tmpa_tl
3561   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3562   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3563   \tl_put_right:Nn \l_tmpa_tl
3564   {
3565     *
3566     {
3567       \int_case:nnF \l_@@_last_col_int
3568       {
3569         { -2 } { \c@MaxMatrixCols }
3570         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3571       }
3572       { \int_eval:n { \l_@@_last_col_int - 1 } }
3573     }
3574     { #2 }
3575   }
3576   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3577   \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3578 }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3579 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3580 \clist_map_inline:nn { p , b , B , v , V }
3581 {
3582   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3583   {
3584     \bool_gset_true:N \g_@@_delims_bool
3585     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3586     % added 2023/10/01
3587     \int_if_zero:nT \l_@@_last_col_int
3588     {
3589       \bool_set_true:N \l_@@_last_col_without_value_bool
3590       \int_set:Nn \l_@@_last_col_int { -1 }
3591     }
3592     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3593     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3594   }
3595   { \use:c { end #1 NiceArray } }
3596 }

```

We define also an environment {NiceMatrix}

```

3597 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3598 {
3599   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3600   % added 2023/10/01
3601   \int_if_zero:nT \l_@@_last_col_int
3602   {
3603     \bool_set_true:N \l_@@_last_col_without_value_bool
3604     \int_set:Nn \l_@@_last_col_int { -1 }
3605   }
3606   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3607   \bool_lazy_or:nnT
3608   { \clist_if_empty_p:N \l_@@_vlines_clist }
3609   { \l_@@_except_borders_bool }
3610   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3611   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3612 }
3613 { \endNiceArray }

```

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

```

3614 \cs_new_protected:Npn \@@_NotEmpty:
3615 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3616 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3617 {

```

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.

```

3618   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3619   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3620   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3621   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3622   \tl_if_empty:NF \l_@@_short_caption_tl
3623   {
3624     \tl_if_empty:NT \l_@@_caption_tl
3625     {
3626       \@@_error_or_warning:n { short-caption-without-caption }
3627       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3628     }
3629   }

```

```

3630 \tl_if_empty:NF \l_@@_label_tl
3631 {
3632   \tl_if_empty:NT \l_@@_caption_tl
3633   { \@@_error_or_warning:n { label-without~caption } }
3634 }
3635 \NewDocumentEnvironment { TabularNote } { b }
3636 {
3637   \bool_if:NTF \l_@@_in_code_after_bool
3638   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3639   {
3640     \tl_if_empty:NF \g_@@_tabularnote_tl
3641     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3642     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3643   }
3644 }
3645 { }
3646 \bool_set_true:N \l_@@_tabular_bool
3647 \NiceArray { #2 }
3648 }
3649 { \endNiceArray }

3650 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3651 {
3652   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3653   \dim_zero_new:N \l_@@_width_dim
3654   \dim_set:Nn \l_@@_width_dim { #1 }
3655   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3656   \bool_set_true:N \l_@@_tabular_bool
3657   \NiceArray { #3 }
3658 }
3659 {
3660   \endNiceArray
3661   \int_if_zero:nT \g_@@_total_X_weight_int
3662   { \@@_error:n { NiceTabularX~without~X } }
3663 }

3664 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3665 {
3666   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3667   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3668   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3669   \bool_set_true:N \l_@@_tabular_bool
3670   \NiceArray { #3 }
3671 }
3672 { \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`).

```

3673 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3674 {
3675   \bool_lazy_all:nT
3676   {
3677     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3678     \l_@@_hvlines_bool
3679     { ! \g_@@_delims_bool }
3680     { ! \l_@@_except_borders_bool }

```

```

3681     }
3682     {
3683         \bool_set_true:N \l_@@_except_borders_bool
3684         \clist_if_empty:NF \l_@@_corners_clist
3685         { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3686         \tl_gput_right:Nn \g_@@_pre_code_after_tl
3687         {
3688             \@@_stroke_block:nnn
3689             {
3690                 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3691                 draw = \l_@@_rules_color_tl
3692             }
3693             { 1-1 }
3694             { \int_use:N \c@iRow - \int_use:N \c@jCol }
3695         }
3696     }
3697 }

```

```

3698 \cs_new_protected:Npn \@@_after_array:
3699 {
3700     \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.

```

3701     \bool_if:NT \g_@@_last_col_found_bool
3702     { \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`.

```

3703     \bool_if:NT \l_@@_last_col_without_value_bool
3704     { \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.

```

3705     \bool_if:NT \l_@@_last_row_without_value_bool
3706     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3707     \tl_build_gput_right:Nx \g_@@_aux_tl
3708     {
3709         \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3710         {
3711             \int_use:N \l_@@_first_row_int ,
3712             \int_use:N \c@iRow ,
3713             \int_use:N \g_@@_row_total_int ,
3714             \int_use:N \l_@@_first_col_int ,
3715             \int_use:N \c@jCol ,
3716             \int_use:N \g_@@_col_total_int
3717         }
3718     }

```

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`).

```

3719     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3720     {
3721         \tl_build_gput_right:Nx \g_@@_aux_tl
3722         {
3723             \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3724             { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3725         }

```

```

3726     }
3727     \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3728     {
3729         \tl_build_gput_right:Nx \g_@@_aux_tl
3730         {
3731             \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3732             { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3733             \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3734             { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3735         }
3736     }

```

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

```

3737     \@@_create_diag_nodes:

```

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

```

3738     \pgfpicture
3739     \int_step_inline:nn \c@iRow
3740     {
3741         \pgfnodealias
3742         { \@@_env: - ##1 - last }
3743         { \@@_env: - ##1 - \int_use:N \c@jCol }
3744     }
3745     \int_step_inline:nn \c@jCol
3746     {
3747         \pgfnodealias
3748         { \@@_env: - last - ##1 }
3749         { \@@_env: - \int_use:N \c@iRow - ##1 }
3750     }
3751     \str_if_empty:NF \l_@@_name_str
3752     {
3753         \int_step_inline:nn \c@iRow
3754         {
3755             \pgfnodealias
3756             { \l_@@_name_str - ##1 - last }
3757             { \@@_env: - ##1 - \int_use:N \c@jCol }
3758         }
3759         \int_step_inline:nn \c@jCol
3760         {
3761             \pgfnodealias
3762             { \l_@@_name_str - last - ##1 }
3763             { \@@_env: - \int_use:N \c@iRow - ##1 }
3764         }
3765     }
3766     \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.

```

3767     \bool_if:NT \l_@@_parallelize_diags_bool
3768     {
3769         \int_gzero_new:N \g_@@_ddots_int
3770         \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.

```

3771         \dim_gzero_new:N \g_@@_delta_x_one_dim
3772         \dim_gzero_new:N \g_@@_delta_y_one_dim
3773         \dim_gzero_new:N \g_@@_delta_x_two_dim

```

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

```

3774     \dim_gzero_new:N \g_@@_delta_y_two_dim
3775   }

3776   \int_zero_new:N \l_@@_initial_i_int
3777   \int_zero_new:N \l_@@_initial_j_int
3778   \int_zero_new:N \l_@@_final_i_int
3779   \int_zero_new:N \l_@@_final_j_int
3780   \bool_set_false:N \l_@@_initial_open_bool
3781   \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.

```

3782   \bool_if:NT \l_@@_small_bool
3783   {
3784     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3785     \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.

```

3786     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3787     { 0.6 \l_@@_xdots_shorten_start_dim }
3788     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3789     { 0.6 \l_@@_xdots_shorten_end_dim }
3790   }

```

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

```

3791   \@@_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.

```

3792   \@@_compute_corners:

```

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

```

3793   \@@_adjust_pos_of_blocks_seq:
3794   \@@_deal_with_rounded_corners:
3795   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3796   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3797   \IfPackageLoadedTF { tikz }
3798   {
3799     \tikzset
3800     {
3801       every~picture / .style =
3802       {
3803         overlay ,
3804         remember~picture ,
3805         name~prefix = \@@_env: -
3806       }
3807     }
3808   }
3809   { }
3810   \cs_set_eq:NN \ialign \@@_old_ialign:
3811   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3812   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3813   \cs_set_eq:NN \OverBrace \@@_OverBrace
3814   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3815   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell

```

```

3816 \cs_set_eq:NN \line \@@_line
3817 \g_@@_pre_code_after_tl
3818 \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.

```

3819 \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).

```

3820 \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`).

```

3821 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3822 { \@@_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:`.

```

3823 \bool_set_true:N \l_@@_in_code_after_bool
3824 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3825 \scan_stop:
3826 \tl_gclear:N \g_nicematrix_code_after_tl
3827 \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.

```

3828 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3829 \tl_if_empty:NF \g_@@_pre_code_before_tl
3830 {
3831   \tl_build_gput_right:Nx \g_@@_aux_tl
3832   {
3833     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3834     { \exp_not:V \g_@@_pre_code_before_tl }
3835   }
3836   \tl_gclear:N \g_@@_pre_code_before_tl
3837 }
3838 \tl_if_empty:NF \g_nicematrix_code_before_tl
3839 {
3840   \tl_build_gput_right:Nx \g_@@_aux_tl
3841   {
3842     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3843     { \exp_not:V \g_nicematrix_code_before_tl }
3844   }
3845   \tl_gclear:N \g_nicematrix_code_before_tl
3846 }
3847 \str_gclear:N \g_@@_name_env_str
3848 \@@_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`.

```

3849 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3850 }

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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`.

```

3851 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3852 { \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).

```

3853 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3854 {
3855   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3856   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3857 }

```

The following command must *not* be protected.

```

3858 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3859 {
3860   { #1 }
3861   { #2 }
3862   {
3863     \int_compare:nNnTF { #3 } > { 99 }
3864     { \int_use:N \c@iRow }
3865     { #3 }
3866   }
3867   {
3868     \int_compare:nNnTF { #4 } > { 99 }
3869     { \int_use:N \c@jCol }
3870     { #4 }
3871   }
3872   { #5 }
3873 }

```

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).

```

3874 \hook_gput_code:nnn { begindocument } { . }
3875 {
3876   \cs_new_protected:Npx \@@_draw_dotted_lines:
3877   {
3878     \c_@@_pgfortikzpicture_tl
3879     \@@_draw_dotted_lines_i:
3880     \c_@@_endpgfortikzpicture_tl
3881   }
3882 }

```

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

```

3883 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3884 {
3885   \pgfrememberpicturepositiononpagetrue
3886   \pgf@relevantforpicturesizefalse
3887   \g_@@_HVdotsfor_lines_tl
3888   \g_@@_Vdots_lines_tl
3889   \g_@@_Ddots_lines_tl
3890   \g_@@_Iddots_lines_tl
3891   \g_@@_Cdots_lines_tl
3892   \g_@@_Ldots_lines_tl
3893 }

```



```

3894 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3895 {
3896   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3897   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3898 }

```

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

```

3899 \pgfdeclareshape { @@_diag_node }
3900 {
3901   \savedanchor { \five }
3902   {
3903     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3904     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3905   }
3906   \anchor { 5 } { \five }
3907   \anchor { center } { \pgfpointorigin }
3908 }

```

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).

```

3909 \cs_new_protected:Npn \@@_create_diag_nodes:
3910 {
3911   \pgfpicture
3912   \pgfrememberpicturepositiononpagetrue
3913   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3914   {
3915     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3916     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3917     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3918     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3919     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3920     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3921     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3922     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3923     \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.

```

3924     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3925     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3926     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3927     \str_if_empty:NF \l_@@_name_str
3928     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3929   }

```

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

```

3930   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3931   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3932   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3933   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3934   \pgfcoordinate
3935   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3936   \pgfnodealias
3937   { \@@_env: - last }
3938   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3939   \str_if_empty:NF \l_@@_name_str
3940   {
3941     \pgfnodealias
3942     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3943     { \@@_env: - \int_use:N \l_tmpa_int }
3944     \pgfnodealias

```

```

3945         { \l_@@_name_str - last }
3946         { \@@_env: - last }
3947     }
3948 \endpgfpicture
3949 }

```

17 We draw the dotted lines

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

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

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

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

This command computes:

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

```

3950 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3951 {

```

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

```

3952 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

3953 \int_set:Nn \l_@@_initial_i_int { #1 }
3954 \int_set:Nn \l_@@_initial_j_int { #2 }
3955 \int_set:Nn \l_@@_final_i_int { #1 }
3956 \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.

```

3957 \bool_set_false:N \l_@@_stop_loop_bool
3958 \bool_do_until:Nn \l_@@_stop_loop_bool
3959 {
3960     \int_add:Nn \l_@@_final_i_int { #3 }
3961     \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3962     \bool_set_false:N \l_@@_final_open_bool
3963     \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3964     {
3965         \int_compare:nNnTF { #3 } = 1
3966         { \bool_set_true:N \l_@@_final_open_bool }
3967         {
3968             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3969             { \bool_set_true:N \l_@@_final_open_bool }
3970         }
3971     }
3972     {
3973         \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3974         {
3975             \int_compare:nNnT { #4 } = { -1 }
3976             { \bool_set_true:N \l_@@_final_open_bool }
3977         }
3978         {
3979             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3980             {
3981                 \int_compare:nNnT { #4 } = 1
3982                 { \bool_set_true:N \l_@@_final_open_bool }
3983             }
3984         }
3985     }
3986     \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.

```

3987     {

```

We do a step backwards.

```

3988         \int_sub:Nn \l_@@_final_i_int { #3 }
3989         \int_sub:Nn \l_@@_final_j_int { #4 }
3990         \bool_set_true:N \l_@@_stop_loop_bool
3991     }

```

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`.

```

3992     {
3993         \cs_if_exist:cTF
3994         {
3995             @@ _ dotted _
3996             \int_use:N \l_@@_final_i_int -
3997             \int_use:N \l_@@_final_j_int
3998         }
3999         {
4000             \int_sub:Nn \l_@@_final_i_int { #3 }
4001             \int_sub:Nn \l_@@_final_j_int { #4 }
4002             \bool_set_true:N \l_@@_final_open_bool
4003             \bool_set_true:N \l_@@_stop_loop_bool
4004         }
4005         {
4006             \cs_if_exist:cTF
4007             {
4008                 pgf @ sh @ ns @ \@@_env:
4009                 - \int_use:N \l_@@_final_i_int
4010                 - \int_use:N \l_@@_final_j_int
4011             }
4012             { \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.

```

4013         {
4014             \cs_set:cpn
4015             {
4016                 @@ _ dotted _
4017                 \int_use:N \l_@@_final_i_int -
4018                 \int_use:N \l_@@_final_j_int
4019             }
4020             { }
4021         }
4022     }
4023 }
4024 }

```

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

```

4025 \bool_set_false:N \l_@@_stop_loop_bool
4026 \bool_do_until:Nn \l_@@_stop_loop_bool
4027 {
4028     \int_sub:Nn \l_@@_initial_i_int { #3 }
4029     \int_sub:Nn \l_@@_initial_j_int { #4 }
4030     \bool_set_false:N \l_@@_initial_open_bool
4031     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4032     {
4033         \int_compare:nNnTF { #3 } = 1
4034         { \bool_set_true:N \l_@@_initial_open_bool }
4035         {
4036             \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4037             { \bool_set_true:N \l_@@_initial_open_bool }
4038         }
4039     }
4040     {
4041         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4042         {
4043             \int_compare:nNnT { #4 } = 1
4044             { \bool_set_true:N \l_@@_initial_open_bool }
4045         }
4046         {
4047             \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4048             {
4049                 \int_compare:nNnT { #4 } = { -1 }
4050                 { \bool_set_true:N \l_@@_initial_open_bool }
4051             }
4052         }
4053     }
4054     \bool_if:NTF \l_@@_initial_open_bool
4055     {
4056         \int_add:Nn \l_@@_initial_i_int { #3 }
4057         \int_add:Nn \l_@@_initial_j_int { #4 }
4058         \bool_set_true:N \l_@@_stop_loop_bool
4059     }
4060     {
4061         \cs_if_exist:cTF
4062         {
4063             @@ _ dotted _
4064             \int_use:N \l_@@_initial_i_int -
4065             \int_use:N \l_@@_initial_j_int
4066         }
4067         {
4068             \int_add:Nn \l_@@_initial_i_int { #3 }
4069             \int_add:Nn \l_@@_initial_j_int { #4 }
4070             \bool_set_true:N \l_@@_initial_open_bool
4071             \bool_set_true:N \l_@@_stop_loop_bool

```

```

4072     }
4073     {
4074         \cs_if_exist:cTF
4075         {
4076             pgf @ sh @ ns @ \l_@@_env:
4077             - \int_use:N \l_@@_initial_i_int
4078             - \int_use:N \l_@@_initial_j_int
4079         }
4080         { \bool_set_true:N \l_@@_stop_loop_bool }
4081         {
4082             \cs_set:cpn
4083             {
4084                 @@ _ dotted _
4085                 \int_use:N \l_@@_initial_i_int -
4086                 \int_use:N \l_@@_initial_j_int
4087             }
4088             { }
4089         }
4090     }
4091 }
4092 }

```

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.

```

4093 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4094 {
4095     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4096     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4097     { \int_use:N \l_@@_final_i_int }
4098     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4099     { } % for the name of the block
4100 }
4101 }

```

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.

```

4102 \cs_new_protected:Npn \l_@@_open_shorten:
4103 {
4104     \bool_if:NT \l_@@_initial_open_bool
4105     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4106     \bool_if:NT \l_@@_final_open_bool
4107     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4108 }

```

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).

```

4109 \cs_new_protected:Npn \l_@@_adjust_to_submatrix:nn #1 #2
4110 {
4111     \int_set:Nn \l_@@_row_min_int 1
4112     \int_set:Nn \l_@@_col_min_int 1
4113     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4114     \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`.

```

4115 \seq_map_inline:Nn \g_@@_submatrix_seq
4116 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4117 }

```

`#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.

```

4118 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4119 {
4120   \bool_if:nT
4121   {
4122     \int_compare_p:n { #3 <= #1 }
4123     && \int_compare_p:n { #1 <= #5 }
4124     && \int_compare_p:n { #4 <= #2 }
4125     && \int_compare_p:n { #2 <= #6 }
4126   }
4127   {
4128     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4129     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4130     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4131     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4132   }
4133 }

```

```

4134 \cs_new_protected:Npn \@@_set_initial_coords:
4135 {
4136   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4137   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4138 }
4139 \cs_new_protected:Npn \@@_set_final_coords:
4140 {
4141   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4142   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4143 }
4144 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4145 {
4146   \pgfpointanchor
4147   {
4148     \@@_env:
4149     - \int_use:N \l_@@_initial_i_int
4150     - \int_use:N \l_@@_initial_j_int
4151   }
4152   { #1 }
4153   \@@_set_initial_coords:
4154 }
4155 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4156 {
4157   \pgfpointanchor
4158   {
4159     \@@_env:
4160     - \int_use:N \l_@@_final_i_int
4161     - \int_use:N \l_@@_final_j_int
4162   }
4163   { #1 }
4164   \@@_set_final_coords:
4165 }
4166 \cs_new_protected:Npn \@@_open_x_initial_dim:
4167 {
4168   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4169   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int

```

```

4170 {
4171   \cs_if_exist:cT
4172   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4173   {
4174     \pgfpointanchor
4175     { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4176     { west }
4177     \dim_set:Nn \l_@@_x_initial_dim
4178     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4179   }
4180 }

```

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

```

4181   \dim_compare:nNtT \l_@@_x_initial_dim = \c_max_dim
4182   {
4183     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4184     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4185     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4186   }
4187 }
4188 \cs_new_protected:Npn \@@_open_x_final_dim:
4189 {
4190   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4191   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4192   {
4193     \cs_if_exist:cT
4194     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4195     {
4196       \pgfpointanchor
4197       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4198       { east }
4199       \dim_set:Nn \l_@@_x_final_dim
4200       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4201     }
4202   }

```

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

```

4203   \dim_compare:nNtT \l_@@_x_final_dim = { - \c_max_dim }
4204   {
4205     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4206     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4207     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4208   }
4209 }

```

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.

```

4210 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4211 {
4212   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4213   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4214   {
4215     \@@_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.

```

4216   \group_begin:
4217   \@@_open_shorten:
4218   \int_if_zero:nTF { #1 }
4219   { \color { nicematrix-first-row } }
4220   {

```

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.

```

4221         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4222         { \color { nicematrix-last-row } }
4223     }
4224     \keys_set:nn { NiceMatrix / xdots } { #3 }
4225     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4226     \@@_actually_draw_Ldots:
4227 \group_end:
4228 }
4229 }
```

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`.

```

4230 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4231 {
4232     \bool_if:NTF \l_@@_initial_open_bool
4233     {
4234         \@@_open_x_initial_dim:
4235         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4236         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4237     }
4238     { \@@_set_initial_coords_from_anchor:n { base~east } }
4239     \bool_if:NTF \l_@@_final_open_bool
4240     {
4241         \@@_open_x_final_dim:
4242         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4243         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4244     }
4245     { \@@_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.

```

4246     \bool_lazy_all:nTF
4247     {
4248         \l_@@_initial_open_bool
4249         \l_@@_final_open_bool
4250         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4251     }
4252     {
4253         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4254         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4255     }
```

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 (?).

```

4256     {
4257         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4258         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4259     }
4260     \@@_draw_line:
4261 }
```


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.

```

4262 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4263 {
4264   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4265   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4266   {
4267     \@@_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.

```

4268     \group_begin:
4269     \@@_open_shorten:
4270     \int_if_zero:nTF { #1 }
4271     { \color { nicematrix-first-row } }
4272     {

```

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.

```

4273         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4274         { \color { nicematrix-last-row } }
4275     }
4276     \keys_set:nn { NiceMatrix / xdots } { #3 }
4277     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4278     \@@_actually_draw_Cdots:
4279   \group_end:
4280 }
4281 }

```

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`.

```

4282 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4283 {
4284   \bool_if:NTF \l_@@_initial_open_bool
4285   { \@@_open_x_initial_dim: }
4286   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4287   \bool_if:NTF \l_@@_final_open_bool
4288   { \@@_open_x_final_dim: }
4289   { \@@_set_final_coords_from_anchor:n { mid-west } }
4290   \bool_lazy_and:nnTF
4291   \l_@@_initial_open_bool
4292   \l_@@_final_open_bool
4293   {
4294     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4295     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4296     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4297     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4298     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4299   }
4300   {
4301     \bool_if:NT \l_@@_initial_open_bool
4302     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4303     \bool_if:NT \l_@@_final_open_bool
4304     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }

```

```

4305     }
4306     \@@_draw_line:
4307 }
4308 \cs_new_protected:Npn \@@_open_y_initial_dim:
4309 {
4310     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4311     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4312     {
4313         \cs_if_exist:cT
4314         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4315         {
4316             \pgfpointanchor
4317             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4318             { north }
4319             \dim_set:Nn \l_@@_y_initial_dim
4320             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4321         }
4322     }
4323     % modified 2023-08-10
4324     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4325     {
4326         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4327         \dim_set:Nn \l_@@_y_initial_dim
4328         {
4329             \fp_to_dim:n
4330             {
4331                 \pgf@y
4332                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4333             }
4334         }
4335     }
4336 }
4337 \cs_new_protected:Npn \@@_open_y_final_dim:
4338 {
4339     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4340     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4341     {
4342         \cs_if_exist:cT
4343         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4344         {
4345             \pgfpointanchor
4346             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4347             { south }
4348             \dim_set:Nn \l_@@_y_final_dim
4349             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4350         }
4351     }
4352     % modified 2023-08-10
4353     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4354     {
4355         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4356         \dim_set:Nn \l_@@_y_final_dim
4357         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4358     }
4359 }

```

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.

```

4360 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4361 {
4362     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4363     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4364     {

```

```

4365 \@@_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.

```

4366 \group_begin:
4367 \@@_open_shorten:
4368 \int_if_zero:nTF { #2 }
4369 { \color { nicematrix-first-col } }
4370 {
4371 \int_compare:nNnT { #2 } = \l_@@_last_col_int
4372 { \color { nicematrix-last-col } }
4373 }
4374 \keys_set:nn { NiceMatrix / xdots } { #3 }
4375 \tl_if_empty:VF \l_@@_xdots_color_tl
4376 { \color { \l_@@_xdots_color_tl } }
4377 \@@_actually_draw_Vdots:
4378 \group_end:
4379 }
4380 }

```

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`.

```

4381 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4382 {

```

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

```

4383 \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.

```

4384 {
4385 \@@_open_y_initial_dim:
4386 \@@_open_y_final_dim:
4387 \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4388 {
4389 \@@_qpoint:n { col - 1 }
4390 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4391 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4392 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4393 % \bool_if:NT \g_@@_delims_bool
4394 % {
4395 \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4396 % }
4397 }
4398 {
4399 \bool_lazy_and:nnTF
4400 { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4401 { \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”.

```

4402     {
4403         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4404         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4405         \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4406         \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4407         % \bool_if:NT \g_@@_delims_bool
4408         % {
4409             \dim_add:Nn
4410                 \l_@@_x_initial_dim
4411                 \c_@@_shift_exterior_Vdots_dim
4412         % }
4413     }

```

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

```

4414     {
4415         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4416         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4417         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4418         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4419     }
4420 }
4421 }

```

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.

```

4422 {
4423     \bool_set_false:N \l_tmpa_bool
4424     \bool_lazy_and:nnT
4425         { ! \l_@@_initial_open_bool }
4426         { ! \l_@@_final_open_bool }
4427     {
4428         \@@_set_initial_coords_from_anchor:n { south-west }
4429         \@@_set_final_coords_from_anchor:n { north-west }
4430         \bool_set:Nn \l_tmpa_bool
4431             { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4432     }

```

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

```

4433     \bool_if:NTF \l_@@_initial_open_bool
4434     {
4435         \@@_open_y_initial_dim:
4436         \@@_set_final_coords_from_anchor:n { north }
4437         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4438     }
4439     {
4440         \@@_set_initial_coords_from_anchor:n { south }
4441         \bool_if:NTF \l_@@_final_open_bool
4442         \@@_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.

```

4443     {
4444         \@@_set_final_coords_from_anchor:n { north }
4445         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4446         {
4447             \dim_set:Nn \l_@@_x_initial_dim
4448             {
4449                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4450                 \l_@@_x_initial_dim \l_@@_x_final_dim
4451             }
4452         }
4453     }
4454 }
4455 }

```

```

4456 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4457 \@@_draw_line:
4458 }

```

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.

```

4459 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4460 {
4461   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4462   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4463   {
4464     \@@_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.

```

4465   \group_begin:
4466   \@@_open_shorten:
4467   \keys_set:nn { NiceMatrix / xdots } { #3 }
4468   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4469   \@@_actually_draw_Ddots:
4470   \group_end:
4471 }
4472 }

```

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`.

```

4473 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4474 {
4475   \bool_if:NTF \l_@@_initial_open_bool
4476   {
4477     \@@_open_y_initial_dim:
4478     \@@_open_x_initial_dim:
4479   }
4480   { \@@_set_initial_coords_from_anchor:n { south-east } }
4481   \bool_if:NTF \l_@@_final_open_bool
4482   {
4483     \@@_open_x_final_dim:
4484     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4485   }
4486   { \@@_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.

```

4487   \bool_if:NT \l_@@_parallelize_diags_bool
4488   {
4489     \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).

```

4490     \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.

```

4491     {
4492         \dim_gset:Nn \g_@@_delta_x_one_dim
4493         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4494         \dim_gset:Nn \g_@@_delta_y_one_dim
4495         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4496     }

```

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`.

```

4497     {
4498         \dim_set:Nn \l_@@_y_final_dim
4499         {
4500             \l_@@_y_initial_dim +
4501             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4502             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4503         }
4504     }
4505 }
4506 \@@_draw_line:
4507 }

```

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.

```

4508 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4509 {
4510     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4511     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4512     {
4513         \@@_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.

```

4514         \group_begin:
4515         \@@_open_shorten:
4516         \keys_set:nn { NiceMatrix / xdots } { #3 }
4517         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4518         \@@_actually_draw_Iddots:
4519         \group_end:
4520     }
4521 }

```

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`.

```

4522 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4523 {
4524     \bool_if:NTF \l_@@_initial_open_bool
4525     {
4526         \@@_open_y_initial_dim:

```

```

4527     \l_@@_open_x_initial_dim:
4528   }
4529   { \l_@@_set_initial_coords_from_anchor:n { south~west } }
4530   \bool_if:NTF \l_@@_final_open_bool
4531   {
4532     \l_@@_open_y_final_dim:
4533     \l_@@_open_x_final_dim:
4534   }
4535   { \l_@@_set_final_coords_from_anchor:n { north~east } }
4536   \bool_if:NT \l_@@_parallelize_diags_bool
4537   {
4538     \int_gincr:N \g_@@_iddots_int
4539     \int_compare:nNnTF \g_@@_iddots_int = 1
4540     {
4541       \dim_gset:Nn \g_@@_delta_x_two_dim
4542       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4543       \dim_gset:Nn \g_@@_delta_y_two_dim
4544       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4545     }
4546     {
4547       \dim_set:Nn \l_@@_y_final_dim
4548       {
4549         \l_@@_y_initial_dim +
4550         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4551         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4552       }
4553     }
4554   }
4555   \l_@@_draw_line:
4556 }

```

18 The actual instructions for drawing the dotted lines with Tikz

The command `\l_@@_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`

```

4557 \cs_new_protected:Npn \l_@@_draw_line:
4558 {
4559   \pgfrememberpicturerepositiononpagetrue
4560   \pgf@relevantforpicturesizefalse
4561   \bool_lazy_or:nnTF
4562   { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4563   \l_@@_dotted_bool
4564   \l_@@_draw_standard_dotted_line:
4565   \l_@@_draw_unstandard_dotted_line:
4566 }

```

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.

```

4567 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4568 {
4569   \begin { scope }
4570   \@@_draw_unstandard_dotted_line:o
4571   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4572 }

```

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.

```

4573 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4574 {
4575   \@@_draw_unstandard_dotted_line:nVVV
4576   { #1 }
4577   \l_@@_xdots_up_tl
4578   \l_@@_xdots_down_tl
4579   \l_@@_xdots_middle_tl
4580 }
4581 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4582 \hook_gput_code:nnn { begindocument } { . }
4583 {
4584   \IfPackageLoadedTF { tikz }
4585   {
4586     \tikzset
4587     {
4588       @@_node_above / .style = { sloped , above } ,
4589       @@_node_below / .style = { sloped , below } ,
4590       @@_node_middle / .style =
4591       {
4592         sloped ,
4593         inner~sep = \c_@@_innersep_middle_dim
4594       }
4595     }
4596   }
4597   { }
4598 }

```

```

4599 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4600 {

```

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.

```

4601 \dim_zero_new:N \l_@@_l_dim
4602 \dim_set:Nn \l_@@_l_dim
4603 {
4604   \fp_to_dim:n
4605   {
4606     sqrt
4607     (
4608       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4609       +
4610       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4611     )

```



```

4612     }
4613   }
4614   \bool_lazy_and:nnT % security
4615   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } < \c_@@_max_l_dim }
4616   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > { 1 pt } }
4617   {
4618     \dim_set:Nn \l_tmpa_dim
4619     {
4620       \l_@@_x_initial_dim
4621       + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4622       * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4623     }
4624     \dim_set:Nn \l_tmpb_dim
4625     {
4626       \l_@@_y_initial_dim
4627       + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4628       * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4629     }
4630     \dim_set:Nn \l_@@_tmpc_dim
4631     {
4632       \l_@@_x_final_dim
4633       - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4634       * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4635     }
4636     \dim_set:Nn \l_@@_tmpd_dim
4637     {
4638       \l_@@_y_final_dim
4639       - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4640       * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4641     }
4642     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4643     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4644     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4645     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4646   }

```

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

```

4647   \bool_if:NT \l_@@_xdots_h_labels_bool
4648   {
4649     \tikzset
4650     {
4651       @@_node_above / .style = { auto = left } ,
4652       @@_node_below / .style = { auto = right } ,
4653       @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4654     }
4655   }
4656   \tl_if_empty:nF { #4 }
4657   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4658   \draw
4659   [ #1 ]
4660   ( \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).

```

4661   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4662   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4663   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4664   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4665   \end { scope }
4666 }
4667 \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).

```

4668 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4669 {
4670   \group_begin:

```

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

```

4671   \dim_zero_new:N \l_@@_l_dim
4672   \dim_set:Nn \l_@@_l_dim
4673   {
4674     \fp_to_dim:n
4675     {
4676       sqrt
4677       (
4678         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4679         +
4680         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4681       )
4682     }
4683   }

```

It seems that, during the first compilations, the value of $\backslash l_@@_l_dim$ 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.

```

4684   \bool_lazy_or:nnF
4685   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4686   { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4687   \@@_draw_standard_dotted_line_i:
4688   \group_end:
4689   \bool_lazy_all:nF
4690   {
4691     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4692     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4693     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4694   }
4695   \l_@@_labels_standard_dotted_line:
4696 }
4697 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4698 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4699 {

```

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

```

4700   \int_set:Nn \l_tmpa_int
4701   {
4702     \dim_ratio:nn
4703     {
4704       \l_@@_l_dim
4705       - \l_@@_xdots_shorten_start_dim
4706       - \l_@@_xdots_shorten_end_dim
4707     }
4708     \l_@@_xdots_inter_dim
4709   }

```

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

```

4710   \dim_set:Nn \l_tmpa_dim
4711   {
4712     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4713     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4714   }
4715   \dim_set:Nn \l_tmpb_dim
4716   {
4717     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *

```

```

4718     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4719 }

```

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.

```

4720 \dim_gadd:Nn \l_@@_x_initial_dim
4721 {
4722   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4723   \dim_ratio:nn
4724   {
4725     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4726     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4727   }
4728   { 2 \l_@@_l_dim }
4729 }
4730 \dim_gadd:Nn \l_@@_y_initial_dim
4731 {
4732   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4733   \dim_ratio:nn
4734   {
4735     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4736     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4737   }
4738   { 2 \l_@@_l_dim }
4739 }
4740 \pgf@relevantforpicturesizefalse
4741 \int_step_inline:nnn 0 \l_tmpa_int
4742 {
4743   \pgfpathcircle
4744   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4745   { \l_@@_xdots_radius_dim }
4746   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4747   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4748 }
4749 \pgfusepathqfill
4750 }

```

```

4751 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4752 {
4753   \pgfscope
4754   \pgftransformshift
4755   {
4756     \pgfpointlineatime { 0.5 }
4757     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4758     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4759   }
4760   \fp_set:Nn \l_tmpa_fp
4761   {
4762     atand
4763     (
4764       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4765       \l_@@_x_final_dim - \l_@@_x_initial_dim
4766     )
4767   }
4768   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4769   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4770   \tl_if_empty:NF \l_@@_xdots_middle_tl
4771   {
4772     \begin { pgfscope }
4773     \pgfset { inner-sep = \c_@@_innersep_middle_dim }
4774     \pgfnode
4775     { rectangle }
4776     { center }

```

```

4777     {
4778         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4779         {
4780             \c_math_toggle_token
4781             \scriptstyle \l_@@_xdots_middle_tl
4782             \c_math_toggle_token
4783         }
4784     }
4785     { }
4786     {
4787         \pgfsetfillcolor { white }
4788         \pgfusepath { fill }
4789     }
4790     \end { pgfscope }
4791 }
4792 \tl_if_empty:NF \l_@@_xdots_up_tl
4793 {
4794     \pgfnode
4795     { rectangle }
4796     { south }
4797     {
4798         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4799         {
4800             \c_math_toggle_token
4801             \scriptstyle \l_@@_xdots_up_tl
4802             \c_math_toggle_token
4803         }
4804     }
4805     { }
4806     { \pgfusepath { } }
4807 }
4808 \tl_if_empty:NF \l_@@_xdots_down_tl
4809 {
4810     \pgfnode
4811     { rectangle }
4812     { north }
4813     {
4814         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4815         {
4816             \c_math_toggle_token
4817             \scriptstyle \l_@@_xdots_down_tl
4818             \c_math_toggle_token
4819         }
4820     }
4821     { }
4822     { \pgfusepath { } }
4823 }
4824 \endpgfscope
4825 }

```

19 User commands available in the new environments

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

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the

catcode is 13 because underscore activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4826 \hook_gput_code:nnn { begindocument } { . }
4827 {
4828   \tl_set:Nn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4829   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4830   \cs_new_protected:Npn \@@_Ldots
4831     { \@@_collect_options:n { \@@_Ldots_i } }
4832   \exp_args:NNV \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4833     {
4834       \int_if_zero:nTF \c@jCol
4835       { \@@_error:nn { in~first~col } \Ldots }
4836       {
4837         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4838         { \@@_error:nn { in~last~col } \Ldots }
4839         {
4840           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4841           { #1 , down = #2 , up = #3 , middle = #4 }
4842         }
4843       }
4844       \bool_if:NF \l_@@_nullify_dots_bool
4845       { \phantom { \ensuremath { \@@_old_ldots } } }
4846       \bool_gset_true:N \g_@@_empty_cell_bool
4847     }

4848   \cs_new_protected:Npn \@@_Cdots
4849     { \@@_collect_options:n { \@@_Cdots_i } }
4850   \exp_args:NNV \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4851     {
4852       \int_if_zero:nTF \c@jCol
4853       { \@@_error:nn { in~first~col } \Cdots }
4854       {
4855         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4856         { \@@_error:nn { in~last~col } \Cdots }
4857         {
4858           \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4859           { #1 , down = #2 , up = #3 , middle = #4 }
4860         }
4861       }
4862       \bool_if:NF \l_@@_nullify_dots_bool
4863       { \phantom { \ensuremath { \@@_old_cdots } } }
4864       \bool_gset_true:N \g_@@_empty_cell_bool
4865     }

4866   \cs_new_protected:Npn \@@_Vdots
4867     { \@@_collect_options:n { \@@_Vdots_i } }
4868   \exp_args:NNV \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4869     {
4870       \int_if_zero:nTF \c@iRow
4871       { \@@_error:nn { in~first~row } \Vdots }
4872       {
4873         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4874         { \@@_error:nn { in~last~row } \Vdots }
4875         {
4876           \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4877           { #1 , down = #2 , up = #3 , middle = #4 }
4878         }
4879       }
4880       \bool_if:NF \l_@@_nullify_dots_bool
4881       { \phantom { \ensuremath { \@@_old_vdots } } }
4882       \bool_gset_true:N \g_@@_empty_cell_bool
4883     }

```

```

4884 \cs_new_protected:Npn \@@_Ddots
4885 { \@@_collect_options:n { \@@_Ddots_i } }
4886 \exp_args:NNV \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4887 {
4888   \int_case:nnF \c@iRow
4889   {
4890     0 { \@@_error:nn { in~first~row } \Ddots }
4891     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4892   }
4893   {
4894     \int_case:nnF \c@jCol
4895     {
4896       0 { \@@_error:nn { in~first~col } \Ddots }
4897       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4898     }
4899     {
4900       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4901       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4902       { #1 , down = #2 , up = #3 , middle = #4 }
4903     }
4904   }
4905 }
4906 \bool_if:NF \l_@@_nullify_dots_bool
4907 { \phantom { \ensuremath { \@@_old_ddots } } }
4908 \bool_gset_true:N \g_@@_empty_cell_bool
4909 }

4910 \cs_new_protected:Npn \@@_Iddots
4911 { \@@_collect_options:n { \@@_Iddots_i } }
4912 \exp_args:NNV \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4913 {
4914   \int_case:nnF \c@iRow
4915   {
4916     0 { \@@_error:nn { in~first~row } \Iddots }
4917     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4918   }
4919   {
4920     \int_case:nnF \c@jCol
4921     {
4922       0 { \@@_error:nn { in~first~col } \Iddots }
4923       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4924     }
4925     {
4926       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4927       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4928       { #1 , down = #2 , up = #3 , middle = #4 }
4929     }
4930   }
4931   \bool_if:NF \l_@@_nullify_dots_bool
4932   { \phantom { \ensuremath { \@@_old_iddots } } }
4933   \bool_gset_true:N \g_@@_empty_cell_bool
4934 }
4935 }

```

End of the \AddToHook.

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

```

4936 \keys_define:nn { NiceMatrix / Ddots }
4937 {
4938   draw-first .bool_set:N = \l_@@_draw_first_bool ,
4939   draw-first .default:n = true ,
4940   draw-first .value_forbidden:n = true
4941 }

```

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

```

4942 \cs_new_protected:Npn \@@_Hspace:
4943 {
4944   \bool_gset_true:N \g_@@_empty_cell_bool
4945   \hspace
4946 }

```

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`.

```

4947 \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`.

```

4948 \cs_new:Npn \@@_Hdotsfor:
4949 {
4950   \bool_lazy_and:nnTF
4951     { \int_if_zero_p:n \c@jCol }
4952     { \int_if_zero_p:n \l_@@_first_col_int }
4953     {
4954       \bool_if:NTF \g_@@_after_col_zero_bool
4955       {
4956         \multicolumn { 1 } { c } { }
4957         \@@_Hdotsfor_i
4958       }
4959       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4960     }
4961     {
4962       \multicolumn { 1 } { c } { }
4963       \@@_Hdotsfor_i
4964     }
4965 }

```

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:`).

```

4966 \hook_gput_code:nnn { begindocument } { . }
4967 {
4968   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4969   \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.

```

4970   \cs_new_protected:Npn \@@_Hdotsfor_i
4971     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
4972   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
4973     {
4974       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4975       {
4976         \@@_Hdotsfor:nnnn
4977         { \int_use:N \c@iRow }
4978         { \int_use:N \c@jCol }
4979         { #2 }
4980         {
4981           #1 , #3 ,
4982           down = \exp_not:n { #4 } ,
4983           up = \exp_not:n { #5 } ,
4984           middle = \exp_not:n { #6 }
4985         }
4986       }
4987       \prg_replicate:nn { #2 - 1 }
4988       {

```

```

4989         &
4990         \multicolumn { 1 } { c } { }
4991         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i: % added 2023-08-26
4992     }
4993 }
4994 }

```

```

4995 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4996 {
4997     \bool_set_false:N \l_@@_initial_open_bool
4998     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4999     \int_set:Nn \l_@@_initial_i_int { #1 }
5000     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5001     \int_compare:nNnTF { #2 } = 1
5002     {
5003         \int_set:Nn \l_@@_initial_j_int 1
5004         \bool_set_true:N \l_@@_initial_open_bool
5005     }
5006     {
5007         \cs_if_exist:cTF
5008         {
5009             pgf @ sh @ ns @ \@@_env:
5010             - \int_use:N \l_@@_initial_i_int
5011             - \int_eval:n { #2 - 1 }
5012         }
5013         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5014         {
5015             \int_set:Nn \l_@@_initial_j_int { #2 }
5016             \bool_set_true:N \l_@@_initial_open_bool
5017         }
5018     }
5019     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5020     {
5021         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5022         \bool_set_true:N \l_@@_final_open_bool
5023     }
5024     {
5025         \cs_if_exist:cTF
5026         {
5027             pgf @ sh @ ns @ \@@_env:
5028             - \int_use:N \l_@@_final_i_int
5029             - \int_eval:n { #2 + #3 }
5030         }
5031         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5032         {
5033             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5034             \bool_set_true:N \l_@@_final_open_bool
5035         }
5036     }
5037
5038     \group_begin:
5039     \@@_open_shorten:
5040     \int_if_zero:nTF { #1 }
5041     {
5042         \color { nicematrix-first-row } }
5043     {
5044         \int_compare:nNnT { #1 } = \g_@@_row_total_int
5045         { \color { nicematrix-last-row } }
5046     }
5047
5048     \keys_set:nn { NiceMatrix / xdots } { #4 }
5049     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }

```



```

5048 \@@_actually_draw_Ldots:
5049 \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).

```

5050 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5051 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5052 }

5053 \hook_gput_code:nnn { begindocument } { . }
5054 {
5055   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5056   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5057   \cs_new_protected:Npn \@@_Vdotsfor:
5058     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5059   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5060     {
5061       \bool_gset_true:N \g_@@_empty_cell_bool
5062       \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5063       {
5064         \@@_Vdotsfor:nnnn
5065         { \int_use:N \c@iRow }
5066         { \int_use:N \c@jCol }
5067         { #2 }
5068         {
5069           #1 , #3 ,
5070           down = \exp_not:n { #4 } ,
5071           up = \exp_not:n { #5 } ,
5072           middle = \exp_not:n { #6 }
5073         }
5074       }
5075     }
5076 }

```

```

5077 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5078 {
5079   \bool_set_false:N \l_@@_initial_open_bool
5080   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5081 \int_set:Nn \l_@@_initial_j_int { #2 }
5082 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5083 \int_compare:nNnTF { #1 } = 1
5084 {
5085   \int_set:Nn \l_@@_initial_i_int 1
5086   \bool_set_true:N \l_@@_initial_open_bool
5087 }
5088 {
5089   \cs_if_exist:cTF
5090   {
5091     pgf @ sh @ ns @ \@@_env:
5092     - \int_eval:n { #1 - 1 }
5093     - \int_use:N \l_@@_initial_j_int
5094   }
5095   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5096   {
5097     \int_set:Nn \l_@@_initial_i_int { #1 }
5098     \bool_set_true:N \l_@@_initial_open_bool
5099   }

```

```

5100     }
5101     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5102     {
5103         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5104         \bool_set_true:N \l_@@_final_open_bool
5105     }
5106     {
5107         \cs_if_exist:cTF
5108         {
5109             pgf @ sh @ ns @ \@@_env:
5110             - \int_eval:n { #1 + #3 }
5111             - \int_use:N \l_@@_final_j_int
5112         }
5113         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5114         {
5115             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5116             \bool_set_true:N \l_@@_final_open_bool
5117         }
5118     }
5119     \group_begin:
5120     \@@_open_shorten:
5121     \int_if_zero:nTF { #2 }
5122     { \color { nicematrix-first-col } }
5123     {
5124         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5125         { \color { nicematrix-last-col } }
5126     }
5127     \keys_set:nn { NiceMatrix / xdots } { #4 }
5128     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5129     \@@_actually_draw_Vdots:
5130     \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).

```

5131     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5132     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5133 }

```

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

```

5134 \NewDocumentCommand \@@_rotate: { 0 { } }
5135 {
5136     \peek_remove_spaces:n
5137     {
5138         \bool_gset_true:N \g_@@_rotate_bool
5139         \keys_set:nn { NiceMatrix / rotate } { #1 }
5140     }
5141 }

5142 \keys_define:nn { NiceMatrix / rotate }
5143 {
5144     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5145     c .value_forbidden:n = true ,
5146     unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
5147 }

```

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

```

5148 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5149 {
5150   \tl_if_empty:nTF { #2 }
5151     { #1 }
5152     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5153 }
5154 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5155 { \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).

```

5156 \hook_gput_code:nnn { begindocument } { . }
5157 {
5158   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5159   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5160   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
5161     {
5162       \group_begin:
5163       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5164       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5165       \use:e
5166       {
5167         \@@_line_i:nn
5168         { \@@_double_int_eval:n #2 - \q_stop }
5169         { \@@_double_int_eval:n #3 - \q_stop }
5170       }
5171       \group_end:
5172     }
5173 }
5174 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5175 {
5176   \bool_set_false:N \l_@@_initial_open_bool
5177   \bool_set_false:N \l_@@_final_open_bool
5178   \bool_if:nTF
5179     {
5180       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
5181       ||
5182       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
5183     }
5184     {
5185       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
5186     }
```

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

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

```

5187     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5188   }

5189 \hook_gput_code:nnn { begindocument } { . }
5190 {
5191   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5192   {

```

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:`.

```

5193     \c_@@_pgfortikzpicture_tl
5194     \@@_draw_line_iii:nn { #1 } { #2 }
5195     \c_@@_endpgfortikzpicture_tl
5196   }
5197 }

```

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

```

5198 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5199 {
5200   \pgfrememberpicturepositiononpagetrue
5201   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5202   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5203   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5204   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5205   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5206   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5207   \@@_draw_line:
5208 }

```

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`

```

5209 \keys_define:nn { NiceMatrix / RowStyle }
5210 {
5211   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5212   cell-space-top-limit .initial:n = \c_zero_dim ,
5213   cell-space-top-limit .value_required:n = true ,
5214   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5215   cell-space-bottom-limit .initial:n = \c_zero_dim ,
5216   cell-space-bottom-limit .value_required:n = true ,
5217   cell-space-limits .meta:n =
5218   {
5219     cell-space-top-limit = #1 ,
5220     cell-space-bottom-limit = #1 ,
5221   } ,
5222   color .tl_set:N = \l_@@_color_tl ,
5223   color .value_required:n = true ,
5224   bold .bool_set:N = \l_tmpa_bool ,
5225   bold .default:n = true ,
5226   bold .initial:n = false ,
5227   nb-rows .code:n =
5228   \str_if_eq:nnTF { #1 } { * }
5229   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5230   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5231   nb-rows .value_required:n = true ,

```

```

5232     rowcolor .tl_set:N = \l_tmpa_tl ,
5233     rowcolor .value_required:n = true ,
5234     rowcolor .initial:n = ,
5235     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5236 }

```

```

5237 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5238 {
5239     \group_begin:
5240     \tl_clear:N \l_tmpa_tl % value of \rowcolor
5241     \tl_clear:N \l_@@_color_tl
5242     \int_set:Nn \l_@@_key_nb_rows_int 1
5243     \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5244     \tl_if_empty:NF \l_tmpa_tl
5245     {

```

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

```

5246         \tl_gput_right:Nx \g_@@_pre_code_before_tl
5247         {

```

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

```

5248             \@@_exp_color_arg:N \@@_rectanglecolor \l_tmpa_tl
5249             { \int_use:N \c@iRow - \int_use:N \c@jCol }
5250             { \int_use:N \c@iRow - * }
5251         }

```

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

```

5252         \int_compare:nNnT \l_@@_key_nb_rows_int > 1
5253         {
5254             \tl_gput_right:Nx \g_@@_pre_code_before_tl
5255             {
5256                 \@@_exp_color_arg:N \@@_rowcolor \l_tmpa_tl
5257                 {
5258                     \int_eval:n { \c@iRow + 1 }
5259                     - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5260                 }
5261             }
5262         }
5263     }
5264     \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
5265     \tl_gput_right:Nx \g_@@_row_style_tl
5266     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5267     \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

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

```

5268     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5269     {
5270         \tl_gput_right:Nx \g_@@_row_style_tl
5271         {
5272             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5273             {
5274                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5275                 { \dim_use:N \l_tmpa_dim }
5276             }
5277         }
5278     }

```

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

```

5279     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5280     {
5281         \tl_gput_right:Nx \g_@@_row_style_tl
5282         {
5283             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5284             {
5285                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim

```

```

5286         { \dim_use:N \l_tmpb_dim }
5287     }
5288 }
5289 }
\l_@@_color_tl is the value of the key color of \RowStyle.
5290 \tl_if_empty:NF \l_@@_color_tl
5291 {
5292     \tl_gput_right:Nx \g_@@_row_style_tl
5293     {
5294         \mode_leave_vertical:
5295         \@@_color:n { \l_@@_color_tl }
5296     }
5297 }
\l_tmpa_bool is the value of the key bold.
5298 \bool_if:NT \l_tmpa_bool
5299 {
5300     \tl_gput_right:Nn \g_@@_row_style_tl
5301     {
5302         \if_mode_math:
5303         \c_math_toggle_token
5304         \bfseries \boldmath
5305         \c_math_toggle_token
5306         \else:
5307         \bfseries \boldmath
5308         \fi:
5309     }
5310 }
5311 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5312 \group_end:
5313 \g_@@_row_style_tl
5314 \ignorespaces
5315 }

```

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).

```

5316 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5317 {

```

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.

```
5318 \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`.

```
5319 \str_if_in:nnF { #1 } { !! }
5320 {
5321   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5322   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5323 }
5324 \int_if_zero:nTF \l_tmpa_int
```

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

```
5325 {
5326   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5327   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5328 }
```

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

```
5329 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5330 }

5331 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5332 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5333 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5334 {
5335   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5336   {
```

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

```
5337   \group_begin:
5338   \pgfsetcornersarced
5339   {
5340     \pgfpoint
5341     { \l_@@_tab_rounded_corners_dim }
5342     { \l_@@_tab_rounded_corners_dim }
5343   }
```

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.

```
5344   \bool_if:NTF \l_@@_hvlines_bool
5345   {
5346     \pgfpathrectanglecorners
5347     {
5348       \pgfpointadd
5349       { \@@_qpoint:n { row-1 } }
5350       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5351     }
5352     {
5353       \pgfpointadd
5354       {
5355         \@@_qpoint:n
5356         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5357       }
5358       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5359     }
5360   }
```

```

5361     {
5362         \pgfpathrectanglecorners
5363         { \@@_qpoint:n { row-1 } }
5364         {
5365             \pgfpointadd
5366             {
5367                 \@@_qpoint:n
5368                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5369             }
5370             { \pgfpoint \c_zero_dim \arrayrulewidth }
5371         }
5372     }
5373     \pgfusepath { clip }
5374     \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5375 }
5376 }

```

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).

```

5377 \cs_new_protected:Npn \@@_actually_color:
5378 {
5379     \pgfpicture
5380     \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.

```

5381     \@@_clip_with_rounded_corners:
5382     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5383     {
5384         \begin { pgfscope }
5385         \@@_color_opacity ##2
5386         \use:c { g_@@_color _ ##1 _tl }
5387         \tl_gclear:c { g_@@_color _ ##1 _tl }
5388         \pgfusepath { fill }
5389         \end { pgfscope }
5390     }
5391     \endpgfpicture
5392 }

```

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

```

5393 \cs_new_protected:Npn \@@_color_opacity
5394 {
5395     \peek_meaning:NTF [
5396         { \@@_color_opacity:w }
5397         { \@@_color_opacity:w [ ] }
5398     }

```

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.

```

5399 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5400 {
5401     \tl_clear:N \l_tmpa_tl
5402     \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.

```

5403     \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5404     \tl_if_empty:NTF \l_tmpb_tl
5405     { \@declaredcolor }
5406     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5407 }

```


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

```

5408 \keys_define:nn { nicematrix / color-opacity }
5409 {
5410   opacity .tl_set:N          = \l_tmpa_tl ,
5411   opacity .value_required:n = true
5412 }

5413 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5414 {
5415   \tl_set:Nn \l_@@_rows_tl { #1 }
5416   \tl_set:Nn \l_@@_cols_tl { #2 }
5417   \@@_cartesian_path:
5418 }

```

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

```

5419 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5420 {
5421   \tl_if_blank:nF { #2 }
5422   {
5423     \@@_add_to_colors_seq:en
5424     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5425     { \@@_cartesian_color:nn { #3 } { - } }
5426   }
5427 }

```

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

```

5428 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5429 {
5430   \tl_if_blank:nF { #2 }
5431   {
5432     \@@_add_to_colors_seq:en
5433     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5434     { \@@_cartesian_color:nn { - } { #3 } }
5435   }
5436 }

```

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

```

5437 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5438 {
5439   \tl_if_blank:nF { #2 }
5440   {
5441     \@@_add_to_colors_seq:en
5442     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5443     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5444   }
5445 }

```

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

```

5446 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5447 {
5448   \tl_if_blank:nF { #2 }
5449   {
5450     \@@_add_to_colors_seq:en
5451     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5452     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5453   }
5454 }

```

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

```

5455 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5456 {
5457   \@@_cut_on_hyphen:w #1 \q_stop
5458   \tl_clear_new:N \l_@@_tmpc_tl
5459   \tl_clear_new:N \l_@@_tmpd_tl
5460   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5461   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5462   \@@_cut_on_hyphen:w #2 \q_stop
5463   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5464   \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`.

```

5465   \@@_cartesian_path:n { #3 }
5466 }

```

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

```

5467 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5468 {
5469   \clist_map_inline:nn { #3 }
5470   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5471 }

```

```

5472 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5473 {
5474   \int_step_inline:nn { \int_use:N \c@iRow }
5475   {
5476     \int_step_inline:nn { \int_use:N \c@jCol }
5477     {
5478       \int_if_even:nTF { ####1 + ##1 }
5479       { \@@_cellcolor [ #1 ] { #2 } }
5480       { \@@_cellcolor [ #1 ] { #3 } }
5481       { ##1 - ####1 }
5482     }
5483   }
5484 }

```

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”.

```

5485 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5486 {
5487   \@@_rectanglecolor [ #1 ] { #2 }
5488   { 1 - 1 }
5489   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5490 }

5491 \keys_define:nn { NiceMatrix / rowcolors }
5492 {
5493   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5494   respect-blocks .default:n = true ,
5495   cols .tl_set:N = \l_@@_cols_tl ,
5496   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5497   restart .default:n = true ,
5498   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5499 }

```

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*.

```
5500 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5501 {
```

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

```
5502 \group_begin:
5503 \seq_clear_new:N \l_@@_colors_seq
5504 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5505 \tl_clear_new:N \l_@@_cols_tl
5506 \tl_set:Nn \l_@@_cols_tl { - }
5507 \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).

```
5508 \int_zero_new:N \l_@@_color_int
5509 \int_set:Nn \l_@@_color_int 1
5510 \bool_if:NT \l_@@_respect_blocks_bool
5511 {
```

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`).

```
5512 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5513 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5514 { \@@_not_in_exterior_p:nnnnn ##1 }
5515 }
5516 \pgfpicture
5517 \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5518 \clist_map_inline:nn { #2 }
5519 {
5520 \tl_set:Nn \l_tmpa_tl { ##1 }
5521 \tl_if_in:NnTF \l_tmpa_tl { - }
5522 { \@@_cut_on_hyphen:w ##1 \q_stop }
5523 { \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.

```
5524 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5525 \int_set:Nn \l_@@_color_int
5526 { \bool_if:NNTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5527 \int_zero_new:N \l_@@_tmpc_int
5528 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5529 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5530 {
```

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

```
5531 \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).

```
5532 \bool_if:NT \l_@@_respect_blocks_bool
5533 {
5534 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5535 { \@@_intersect_our_row_p:nnnnn #####1 }
5536 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }
```

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

```
5537 }
5538 \tl_set:Nx \l_@@_rows_tl
5539 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```

5540         \tl_clear_new:N \l_@@_color_tl
5541         \tl_set:Nx \l_@@_color_tl
5542         {
5543             \@@_color_index:n
5544             {
5545                 \int_mod:nn
5546                 { \l_@@_color_int - 1 }
5547                 { \seq_count:N \l_@@_colors_seq }
5548                 + 1
5549             }
5550         }
5551         \tl_if_empty:NF \l_@@_color_tl
5552         {
5553             \@@_add_to_colors_seq:ee
5554             { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5555             { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5556         }
5557         \int_incr:N \l_@@_color_int
5558         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5559     }
5560 }
5561 \endpgfpicture
5562 \group_end:
5563 }

```

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.

```

5564 \cs_new:Npn \@@_color_index:n #1
5565 {
5566     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5567     { \@@_color_index:n { #1 - 1 } }
5568     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5569 }

```

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 curryfication.

```

5570 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5571 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5572 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5573 {
5574     \int_compare:nNnT { #3 } > \l_tmpb_int
5575     { \int_set:Nn \l_tmpb_int { #3 } }
5576 }

5577 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5578 {
5579     \bool_lazy_or:nnTF
5580     { \int_if_zero_p:n { #4 } }
5581     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5582     \prg_return_false:
5583     \prg_return_true:
5584 }

```

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

```

5585 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5586 {

```

```

5587 \bool_if:nTF
5588 {
5589   \int_compare_p:n { #1 <= \l_tmpa_int }
5590   &&
5591   \int_compare_p:n { \l_tmpa_int <= #3 }
5592 }
5593 \prg_return_true:
5594 \prg_return_false:
5595 }

```

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`).

```

5596 \cs_new_protected:Npn \@@_cartesian_path:n #1
5597 {
5598   \bool_lazy_and:nnT
5599   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5600   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5601   {
5602     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5603     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5604   }

```

We begin the loop over the columns.

```

5605 \clist_map_inline:Nn \l_@@_cols_tl
5606 {
5607   \tl_set:Nn \l_tmpa_tl { ##1 }
5608   \tl_if_in:NnTF \l_tmpa_tl { - }
5609   { \@@_cut_on_hyphen:w ##1 \q_stop }
5610   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5611   \bool_lazy_or:nnT
5612   { \tl_if_blank_p:V \l_tmpa_tl }
5613   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5614   { \tl_set:Nn \l_tmpa_tl { 1 } }
5615   \bool_lazy_or:nnT
5616   { \tl_if_blank_p:V \l_tmpb_tl }
5617   { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5618   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5619   \int_compare:nNnT \l_tmpb_tl > \c@jCol
5620   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

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

```

5621 \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.

```

5622 \@@_qpoint:n { col - \l_tmpa_tl }
5623 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5624 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5625 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5626 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5627 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5628 \clist_map_inline:Nn \l_@@_rows_tl
5629 {
5630   \tl_set:Nn \l_tmpa_tl { #####1 }
5631   \tl_if_in:NnTF \l_tmpa_tl { - }
5632   { \@@_cut_on_hyphen:w #####1 \q_stop }
5633   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }

```

```

5634 \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5635 \tl_if_empty:NT \l_tmpb_tl
5636 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5637 \int_compare:nNnT \l_tmpb_tl > \c@iRow
5638 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
Now, the numbers of both rows are in \l_tmpa_tl and \l_tmpb_tl.
5639 \seq_if_in:NxF \l_@@_corners_cells_seq
5640 { \l_tmpa_tl - \l_@@_tmpc_tl }
5641 {
5642 \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5643 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5644 \@@_qpoint:n { row - \l_tmpa_tl }
5645 \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5646 \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5647 \pgfpathrectanglecorners
5648 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5649 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5650 }
5651 }
5652 }
5653 }

```

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`).

```

5654 \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.

```

5655 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5656 {
5657 \clist_set_eq:NN \l_tmpa_clist #1
5658 \clist_clear:N #1
5659 \clist_map_inline:Nn \l_tmpa_clist
5660 {
5661 \tl_set:Nn \l_tmpa_tl { ##1 }
5662 \tl_if_in:NnTF \l_tmpa_tl { - }
5663 { \@@_cut_on_hyphen:w ##1 \q_stop }
5664 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5665 \bool_lazy_or:nnT
5666 { \tl_if_blank_p:V \l_tmpa_tl }
5667 { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5668 { \tl_set:Nn \l_tmpa_tl { 1 } }
5669 \bool_lazy_or:nnT
5670 { \tl_if_blank_p:V \l_tmpb_tl }
5671 { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5672 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5673 \int_compare:nNnT \l_tmpb_tl > #2
5674 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5675 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5676 { \clist_put_right:Nn #1 { #####1 } }
5677 }
5678 }

```

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

```

5679 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5680 {
5681 \@@_test_color_inside:
5682 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5683 {

```

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`).

```

5684     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5685     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5686   }
5687   \ignorespaces
5688 }

```

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

```

5689 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5690 {
5691   \@@_test_color_inside:
5692   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5693   {
5694     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5695     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5696     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5697   }
5698   \ignorespaces
5699 }

```

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).

```

5700 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5701 { \@@_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.

```

5702 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5703 {
5704   \@@_test_color_inside:
5705   \peek_remove_spaces:n
5706   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5707 }

5708 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5709 {

```

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`.

```

5710   \seq_gclear:N \g_tmpa_seq
5711   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5712   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5713   \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`.

```

5714   \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5715   {
5716     { \int_use:N \c@iRow }
5717     { \exp_not:n { #1 } }
5718     { \exp_not:n { #2 } }
5719     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5720   }
5721 }

```

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`).

```
5722 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5723 {
5724   \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).

```
5725   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5726   {
5727     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5728     {
5729       \@@_rowlistcolors
5730       [ \exp_not:n { #2 } ]
5731       { #1 - \int_eval:n { \c@iRow - 1 } }
5732       { \exp_not:n { #3 } }
5733       [ \exp_not:n { #4 } ]
5734     }
5735   }
5736 }
```

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.

```
5737 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5738 {
5739   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5740   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5741   \seq_gclear:N \g_@@_rowlistcolors_seq
5742 }

5743 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5744 {
5745   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5746   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5747 }
```

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.

```
5748 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5749 {
```

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).

```
5750   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5751   {
```

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).

```
5752     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5753     {
5754       \exp_not:N \columncolor [ #1 ]
5755       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5756     }
5757   }
5758 }
```


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`.

```
5759 \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`.

```
5760 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5761 {
5762   \int_if_zero:nTF \l_@@_first_col_int
5763     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5764     {
5765       \int_if_zero:nTF \c@jCol
5766         {
5767           \int_compare:nNf \c@iRow = { -1 }
5768             { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5769         }
5770       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5771     }
5772 }
```

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.

```
5773 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5774 {
5775   \int_if_zero:nF \c@iRow
5776   {
5777     \int_compare:nNf \c@iRow = \l_@@_last_row_int
5778     {
5779       \int_compare:nNt \c@jCol > \c_zero_int
5780       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5781     }
5782   }
5783 }
```

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.

```
5784 \keys_define:nn { NiceMatrix / Rules }
5785 {
5786   position .int_set:N = \l_@@_position_int ,
5787   position .value_required:n = true ,
5788   start .int_set:N = \l_@@_start_int ,
```

```

5789   start .initial:n = 1 ,
5790   end .code:n =
5791     \bool_lazy_or:nnTF
5792     { \tl_if_empty_p:n { #1 } }
5793     { \str_if_eq_p:nn { #1 } { last } }
5794     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5795     { \int_set:Nn \l_@@_end_int { #1 } }
5796   }

```

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.

```

5797 \keys_define:nn { NiceMatrix / RulesBis }
5798 {
5799   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5800   multiplicity .initial:n = 1 ,
5801   dotted .bool_set:N = \l_@@_dotted_bool ,
5802   dotted .initial:n = false ,
5803   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]`.

```

5804   color .code:n =
5805     \@@_set_CT@arc@:n { #1 }
5806     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5807   color .value_required:n = true ,
5808   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5809   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.

```

5810   tikz .code:n =
5811     \IfPackageLoadedTF { tikz }
5812     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5813     { \@@_error:n { tikz-without~tikz } } ,
5814   tikz .value_required:n = true ,
5815   total-width .dim_set:N = \l_@@_rule_width_dim ,
5816   total-width .value_required:n = true ,
5817   width .meta:n = { total-width = #1 } ,
5818   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5819 }

```

The vertical rules

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

```

5820 \cs_new_protected:Npn \@@_vline:n #1
5821 {

```

The group is for the options.

```

5822   \group_begin:
5823   \int_zero_new:N \l_@@_end_int
5824   \int_set_eq:NN \l_@@_end_int \c@iRow
5825   \keys_set_known:nn { 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).

```

5826 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5827 \@@_vline_i:
5828 \group_end:
5829 }

```

```

5830 \cs_new_protected:Npn \@@_vline_i:
5831 {
5832 \int_zero_new:N \l_@@_local_start_int
5833 \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`.

```

5834 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5835 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5836 \l_tmpa_tl
5837 {

```

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.

```

5838 \bool_gset_true:N \g_tmpa_bool
5839 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5840 { \@@_test_vline_in_block:nnnnn ##1 }
5841 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5842 { \@@_test_vline_in_block:nnnnn ##1 }
5843 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5844 { \@@_test_vline_in_stroken_block:nnnn ##1 }
5845 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5846 \bool_if:NTF \g_tmpa_bool
5847 {
5848 \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.

```

5849 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5850 }
5851 {
5852 \int_compare:nNnT \l_@@_local_start_int > 0
5853 {
5854 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5855 \@@_vline_ii:
5856 \int_zero:N \l_@@_local_start_int
5857 }
5858 }
5859 }
5860 \int_compare:nNnT \l_@@_local_start_int > 0
5861 {
5862 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5863 \@@_vline_ii:
5864 }
5865 }

```

```

5866 \cs_new_protected:Npn \@@_test_in_corner_v:
5867 {
5868 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5869 {
5870 \seq_if_in:NxT
5871 \l_@@_corners_cells_seq
5872 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5873 { \bool_set_false:N \g_tmpa_bool }
5874 }

```

```

5875 {
5876   \seq_if_in:NxT
5877   \l_@@_corners_cells_seq
5878   { \l_tmpa_tl - \l_tmpb_tl }
5879   {
5880     \int_compare:nNnTF \l_tmpb_tl = 1
5881     { \bool_set_false:N \g_tmpa_bool }
5882     {
5883       \seq_if_in:NxT
5884       \l_@@_corners_cells_seq
5885       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5886       { \bool_set_false:N \g_tmpa_bool }
5887     }
5888   }
5889 }
5890 }

```

```

5891 \cs_new_protected:Npn \@@_vline_ii:
5892 {
5893   \tl_clear:N \l_@@_tikz_rule_tl
5894   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5895   \bool_if:NTF \l_@@_dotted_bool
5896   \@@_vline_iv:
5897   {
5898     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5899     \@@_vline_iii:
5900     \@@_vline_v:
5901   }
5902 }

```

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

```

5903 \cs_new_protected:Npn \@@_vline_iii:
5904 {
5905   \pgfpicture
5906   \pgfrememberpicturerepositiononpagetrue
5907   \pgf@relevantforpicturesizefalse
5908   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5909   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5910   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5911   \dim_set:Nn \l_tmpb_dim
5912   {
5913     \pgf@x
5914     - 0.5 \l_@@_rule_width_dim
5915     +
5916     ( \arrayrulewidth * \l_@@_multiplicity_int
5917       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5918   }
5919   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5920   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5921   \bool_lazy_all:nT
5922   {
5923     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5924     { \cs_if_exist_p:N \CT@drsc@ }
5925     { ! \tl_if_blank_p:V \CT@drsc@ }
5926   }
5927   {
5928     \group_begin:
5929     \CT@drsc@
5930     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5931     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5932     \dim_set:Nn \l_@@_tmpd_dim
5933     {

```

```

5934         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5935         * ( \l_@@_multiplicity_int - 1 )
5936     }
5937     \pgfpathrectanglecorners
5938     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5939     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5940     \pgfusepath { fill }
5941     \group_end:
5942 }
5943 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5944 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5945 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5946 {
5947     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5948     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5949     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5950     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5951 }
5952 \CT@arc@
5953 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5954 \pgfsetrectcap
5955 \pgfusepathqstroke
5956 \endpgfpicture
5957 }

```

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

```

5958 \cs_new_protected:Npn \@@_vline_iv:
5959 {
5960     \pgfpicture
5961     \pgfrememberpicturepositiononpagetrue
5962     \pgf@relevantforpicturesizefalse
5963     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5964     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5965     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5966     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5967     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5968     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5969     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5970     \CT@arc@
5971     \@@_draw_line:
5972     \endpgfpicture
5973 }

```

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

```

5974 \cs_new_protected:Npn \@@_vline_v:
5975 {
5976     \begin {tikzpicture }
5977     % 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).

```

5978     \CT@arc@
5979     \tl_if_empty:NF \l_@@_rule_color_tl
5980     { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
5981     \pgfrememberpicturepositiononpagetrue
5982     \pgf@relevantforpicturesizefalse
5983     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5984     \dim_set_eq:NN \l_tmpa_dim \pgf@y
5985     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5986     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5987     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5988     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y

```

```

5989 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5990 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5991 ( \l_tmpb_dim , \l_tmpa_dim ) --
5992 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5993 \end { tikzpicture }
5994 }

```

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).

```

5995 \cs_new_protected:Npn \@@_draw_vlines:
5996 {
5997   \int_step_inline:nnn
5998   {
5999     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6000     1 2
6001   }
6002   {
6003     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6004     { \int_eval:n { \c@jCol + 1 } }
6005     \c@jCol
6006   }
6007   {
6008     \tl_if_eq:NnF \l_@@_vlines_clist { all }
6009     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6010     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6011   }
6012 }

```

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}`.

```

6013 \cs_new_protected:Npn \@@_hline:n #1
6014 {

```

The group is for the options.

```

6015   \group_begin:
6016   \int_zero_new:N \l_@@_end_int
6017   \int_set_eq:NN \l_@@_end_int \c@jCol
6018   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6019   \@@_hline_i:
6020   \group_end:
6021 }

```

```

6022 \cs_new_protected:Npn \@@_hline_i:
6023 {
6024   \int_zero_new:N \l_@@_local_start_int
6025   \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`.

```

6026   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
6027   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6028   \l_tmpb_tl
6029   {

```

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.

```

6030   \bool_gset_true:N \g_tmpa_bool
6031   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6032   { \@@_test_hline_in_block:nnnnn ##1 }

```

```

6033 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6034 { \@@_test_hline_in_block:nnnnn ##1 }
6035 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6036 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6037 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6038 \bool_if:NTF \g_tmpa_bool
6039 {
6040 \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.

```

6041 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6042 }
6043 {
6044 \int_compare:nNnT \l_@@_local_start_int > 0
6045 {
6046 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6047 \@@_hline_ii:
6048 \int_zero:N \l_@@_local_start_int
6049 }
6050 }
6051 }
6052 \int_compare:nNnT \l_@@_local_start_int > 0
6053 {
6054 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6055 \@@_hline_ii:
6056 }
6057 }

```

```

6058 \cs_new_protected:Npn \@@_test_in_corner_h:
6059 {
6060 \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6061 {
6062 \seq_if_in:NxT
6063 \l_@@_corners_cells_seq
6064 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6065 { \bool_set_false:N \g_tmpa_bool }
6066 }
6067 {
6068 \seq_if_in:NxT
6069 \l_@@_corners_cells_seq
6070 { \l_tmpa_tl - \l_tmpb_tl }
6071 {
6072 \int_compare:nNnTF \l_tmpa_tl = 1
6073 { \bool_set_false:N \g_tmpa_bool }
6074 {
6075 \seq_if_in:NxT
6076 \l_@@_corners_cells_seq
6077 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6078 { \bool_set_false:N \g_tmpa_bool }
6079 }
6080 }
6081 }
6082 }

```

```

6083 \cs_new_protected:Npn \@@_hline_ii:
6084 {
6085 \tl_clear:N \l_@@_tikz_rule_tl
6086 \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6087 \bool_if:NTF \l_@@_dotted_bool
6088 \@@_hline_iv:
6089 {

```

```

6090     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6091     \@@_hline_iii:
6092     \@@_hline_v:
6093   }
6094 }

```

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

```

6095 \cs_new_protected:Npn \@@_hline_iii:
6096 {
6097   \pgfpicture
6098   \pgfrememberpicturepositiononpagetrue
6099   \pgf@relevantforpicturesizefalse
6100   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6101   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6102   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6103   \dim_set:Nn \l_tmpb_dim
6104   {
6105     \pgf@y
6106     - 0.5 \l_@@_rule_width_dim
6107     +
6108     ( \arrayrulewidth * \l_@@_multiplicity_int
6109       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6110   }
6111   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6112   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6113   \bool_lazy_all:nT
6114   {
6115     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6116     { \cs_if_exist_p:N \CT@drsc@ }
6117     { ! \tl_if_blank_p:V \CT@drsc@ }
6118   }
6119   {
6120     \group_begin:
6121     \CT@drsc@
6122     \dim_set:Nn \l_@@_tmpd_dim
6123     {
6124       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6125       * ( \l_@@_multiplicity_int - 1 )
6126     }
6127     \pgfpathrectanglecorners
6128     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6129     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6130     \pgfusepathqfill
6131     \group_end:
6132   }
6133   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6134   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6135   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6136   {
6137     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6138     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6139     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6140     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6141   }
6142   \CT@arc@
6143   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6144   \pgfsetrectcap
6145   \pgfusepathqstroke
6146   \endpgfpicture
6147 }

```

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}

```

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

```

6148 \cs_new_protected:Npn \@@_hline_iv:
6149 {
6150   \pgfpicture
6151   \pgfrememberpicturepositiononpagetrue
6152   \pgf@relevantforpicturesizefalse
6153   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6154   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6155   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6156   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6157   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6158   \int_compare:nNnT \l_@@_local_start_int = 1
6159   {
6160     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6161     \bool_if:NF \g_@@_delims_bool
6162     { \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.

```

6163     \tl_if_eq:NnF \g_@@_left_delim_tl (
6164       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6165     )
6166     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6167     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6168     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6169     {
6170       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6171       \bool_if:NF \g_@@_delims_bool
6172       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6173       \tl_if_eq:NnF \g_@@_right_delim_tl (
6174         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6175       )
6176       \CT@arc@
6177       \@@_draw_line:
6178       \endpgfpicture
6179     }

```

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`).

```

6180 \cs_new_protected:Npn \@@_hline_v:
6181 {
6182   \begin { tikzpicture }
6183   % 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).

```

6184   \CT@arc@

```

```

6185 \tl_if_empty:NF \l_@@_rule_color_tl
6186 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6187 \pgfrememberpicturepositiononpagetrue
6188 \pgf@relevantforpicturesizefalse
6189 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6190 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6191 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6192 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6193 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6194 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6195 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6196 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6197 ( \l_tmpa_dim , \l_tmpb_dim ) --
6198 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6199 \end { tikzpicture }
6200 }

```

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).

```

6201 \cs_new_protected:Npn \@@_draw_hlines:
6202 {
6203   \int_step_inline:nnn
6204   {
6205     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6206     1 2
6207   }
6208   {
6209     \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6210     { \int_eval:n { \c@iRow + 1 } }
6211     \c@iRow
6212   }
6213   {
6214     \tl_if_eq:NnF \l_@@_hlines_clist { all }
6215     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6216     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6217   }
6218 }

```

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

```

6219 \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.

```

6220 \cs_set:Npn \@@_Hline_i:n #1
6221 {
6222   \peek_remove_spaces:n
6223   {
6224     \peek_meaning:NTF \Hline
6225     { \@@_Hline_ii:nn { #1 + 1 } }
6226     { \@@_Hline_iii:n { #1 } }
6227   }
6228 }
6229 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6230 \cs_set:Npn \@@_Hline_iii:n #1
6231 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6232 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6233 {
6234   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6235   \skip_vertical:n { \l_@@_rule_width_dim }
6236   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6237   {

```

```

6238 \@@_hline:n
6239 {
6240     multiplicity = #1 ,
6241     position = \int_eval:n { \c@iRow + 1 } ,
6242     total-width = \dim_use:N \l_@@_rule_width_dim ,
6243     #2
6244 }
6245 }
6246 \egroup
6247 }

```

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`).

```

6248 \cs_new_protected:Npn \@@_custom_line:n #1
6249 {
6250     \str_clear_new:N \l_@@_command_str
6251     \str_clear_new:N \l_@@_ccommand_str
6252     \str_clear_new:N \l_@@_letter_str
6253     \tl_clear_new:N \l_@@_other_keys_tl
6254     \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.

```

6255     \bool_lazy_all:nTF
6256     {
6257         { \str_if_empty_p:N \l_@@_letter_str }
6258         { \str_if_empty_p:N \l_@@_command_str }
6259         { \str_if_empty_p:N \l_@@_ccommand_str }
6260     }
6261     { \@@_error:n { No-letter~and-no-command } }
6262     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
6263 }
6264 \keys_define:nn { NiceMatrix / custom-line }
6265 {
6266     letter .str_set:N = \l_@@_letter_str ,
6267     letter .value_required:n = true ,
6268     command .str_set:N = \l_@@_command_str ,
6269     command .value_required:n = true ,
6270     ccommand .str_set:N = \l_@@_ccommand_str ,
6271     ccommand .value_required:n = true ,
6272 }

```

```

6273 \cs_new_protected:Npn \@@_custom_line_i:n #1
6274 {

```

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

```

6275     \bool_set_false:N \l_@@_tikz_rule_bool
6276     \bool_set_false:N \l_@@_dotted_rule_bool
6277     \bool_set_false:N \l_@@_color_bool
6278     \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6279     \bool_if:NT \l_@@_tikz_rule_bool
6280     {
6281         \IfPackageLoadedTF { tikz }
6282         { }

```

```

6283         { \@@_error:n { tikz~in~custom~line~without~tikz } }
6284     \bool_if:NT \l_@@_color_bool
6285         { \@@_error:n { color~in~custom~line~with~tikz } }
6286     }
6287 \bool_if:nT
6288     {
6289         \int_compare_p:nNn \l_@@_multiplicity_int > 1
6290         && \l_@@_dotted_rule_bool
6291     }
6292     { \@@_error:n { key~multiplicity~with~dotted } }
6293 \str_if_empty:NF \l_@@_letter_str
6294     {
6295         \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6296         { \@@_error:n { Several~letters } }
6297         {
6298             \exp_args:NnV \tl_if_in:NnTF
6299             \c_@@_forbidden_letters_str \l_@@_letter_str
6300             { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6301             {

```

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.

```

6302         \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6303         { \@@_v_custom_line:n { #1 } }
6304     }
6305 }
6306 }
6307 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6308 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6309 }
6310 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6311 \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.

```

6312 \keys_define:nn { NiceMatrix / custom-line-bis }
6313 {
6314     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6315     multiplicity .initial:n = 1 ,
6316     multiplicity .value_required:n = true ,
6317     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6318     color .value_required:n = true ,
6319     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6320     tikz .value_required:n = true ,
6321     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6322     dotted .value_forbidden:n = true ,
6323     total-width .code:n = { } ,
6324     total-width .value_required:n = true ,
6325     width .code:n = { } ,
6326     width .value_required:n = true ,
6327     sep-color .code:n = { } ,
6328     sep-color .value_required:n = true ,
6329     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6330 }

```

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

```

6331 \bool_new:N \l_@@_dotted_rule_bool
6332 \bool_new:N \l_@@_tikz_rule_bool
6333 \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`.

```

6334 \keys_define:nn { NiceMatrix / custom-line-width }
6335 {
6336   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6337   multiplicity .initial:n = 1 ,
6338   multiplicity .value_required:n = true ,
6339   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6340   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6341                       \bool_set_true:N \l_@@_total_width_bool ,
6342   total-width .value_required:n = true ,
6343   width .meta:n = { total-width = #1 } ,
6344   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6345 }

```

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`).

```

6346 \cs_new_protected:Npn \@@_h_custom_line:n #1
6347 {

```

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`).

```

6348   \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6349   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6350 }

```

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`).

```

6351 \cs_new_protected:Npn \@@_c_custom_line:n #1
6352 {

```

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

```

6353   \exp_args:Nc \NewExpandableDocumentCommand
6354   { nicematrix - \l_@@_ccommand_str }
6355   { 0 { } m }
6356   {
6357     \noalign
6358     {
6359       \@@_compute_rule_width:n { #1 , ##1 }
6360       \skip_vertical:n { \l_@@_rule_width_dim }
6361       \clist_map_inline:nn
6362       { ##2 }
6363       { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6364     }
6365   }
6366   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6367 }

```

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*.

```

6368 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6369 {
6370   \str_if_in:nnTF { #2 } { - }
6371   { \@@_cut_on_hyphen:w #2 \q_stop }
6372   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6373   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6374   {
6375     \@@_hline:n
6376     {

```

```

6377         #1 ,
6378         start = \l_tmpa_tl ,
6379         end = \l_tmpb_tl ,
6380         position = \int_eval:n { \c@iRow + 1 } ,
6381         total-width = \dim_use:N \l_@@_rule_width_dim
6382     }
6383 }
6384 }

6385 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6386 {
6387     \bool_set_false:N \l_@@_tikz_rule_bool
6388     \bool_set_false:N \l_@@_total_width_bool
6389     \bool_set_false:N \l_@@_dotted_rule_bool
6390     \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6391     \bool_if:NF \l_@@_total_width_bool
6392     {
6393         \bool_if:NTF \l_@@_dotted_rule_bool
6394         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6395         {
6396             \bool_if:NF \l_@@_tikz_rule_bool
6397             {
6398                 \dim_set:Nn \l_@@_rule_width_dim
6399                 {
6400                     \arrayrulewidth * \l_@@_multiplicity_int
6401                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6402                 }
6403             }
6404         }
6405     }
6406 }

6407 \cs_new_protected:Npn \@@_v_custom_line:n #1
6408 {
6409     \@@_compute_rule_width:n { #1 }

```

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

```

6410     \tl_gput_right:Nx \g_@@_array_preamble_tl
6411     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6412     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6413     {
6414         \@@_vline:n
6415         {
6416             #1 ,
6417             position = \int_eval:n { \c@jCol + 1 } ,
6418             total-width = \dim_use:N \l_@@_rule_width_dim
6419         }
6420     }
6421     \@@_rec_preamble:n
6422 }

6423 \@@_custom_line:n
6424 { 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.

```

6425 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6426 {
6427     \bool_lazy_all:nT
6428     {
6429         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }

```

```

6430     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6431     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6432     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6433   }
6434   { \bool_gset_false:N \g_tmpa_bool }
6435 }

```

The same for vertical rules.

```

6436 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6437 {
6438   \bool_lazy_all:nT
6439   {
6440     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6441     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6442     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6443     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6444   }
6445   { \bool_gset_false:N \g_tmpa_bool }
6446 }
6447 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6448 {
6449   \bool_lazy_all:nT
6450   {
6451     {
6452       ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6453       || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6454     }
6455     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6456     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6457   }
6458   { \bool_gset_false:N \g_tmpa_bool }
6459 }
6460 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6461 {
6462   \bool_lazy_all:nT
6463   {
6464     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6465     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6466     {
6467       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6468       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6469     }
6470   }
6471   { \bool_gset_false:N \g_tmpa_bool }
6472 }

```

24 The key corners

When the key `corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```

6473 \cs_new_protected:Npn \@@_compute_corners:
6474 {

```

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.

```

6475   \seq_clear_new:N \l_@@_corners_cells_seq
6476   \clist_map_inline:Nn \l_@@_corners_clist
6477   {

```

```

6478 \str_case:nnF { ##1 }
6479 {
6480   { NW }
6481   { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6482   { NE }
6483   { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6484   { SW }
6485   { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6486   { SE }
6487   { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6488 }
6489 { \@@_error:nn { bad~corner } { ##1 } }
6490 }

```

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

```

6491 \seq_if_empty:NF \l_@@_corners_cells_seq
6492 {

```

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.

```

6493 \tl_build_gput_right:Nx \g_@@_aux_tl
6494 {
6495   \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6496   { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6497 }
6498 }
6499 }

```

“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.

```

6500 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6501 {

```

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.

```

6502 \bool_set_false:N \l_tmpa_bool
6503 \int_zero_new:N \l_@@_last_empty_row_int
6504 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6505 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6506 {
6507   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6508   \bool_lazy_or:nnTF
6509   {
6510     \cs_if_exist_p:c
6511     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6512   }
6513   \l_tmpb_bool
6514   { \bool_set_true:N \l_tmpa_bool }
6515   {
6516     \bool_if:NF \l_tmpa_bool

```



```

6517         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6518     }
6519 }

```

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

```

6520 \bool_set_false:N \l_tmpa_bool
6521 \int_zero_new:N \l_@@_last_empty_column_int
6522 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6523 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6524 {
6525     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6526     \bool_lazy_or:nnTF
6527         \l_tmpb_bool
6528         {
6529             \cs_if_exist_p:c
6530                 { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6531         }
6532         { \bool_set_true:N \l_tmpa_bool }
6533         {
6534             \bool_if:NF \l_tmpa_bool
6535                 { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6536         }
6537     }

```

Now, we loop over the rows.

```

6538 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6539 {

```

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

```

6540     \bool_set_false:N \l_tmpa_bool
6541     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6542     {
6543         \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6544         \bool_lazy_or:nnTF
6545             \l_tmpb_bool
6546             {
6547                 \cs_if_exist_p:c
6548                     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6549             }
6550             { \bool_set_true:N \l_tmpa_bool }
6551             {
6552                 \bool_if:NF \l_tmpa_bool
6553                 {
6554                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6555                     \seq_put_right:Nn
6556                         \l_@@_corners_cells_seq
6557                         { ##1 - #####1 }
6558                 }
6559             }
6560     }
6561 }
6562 }

```

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`).

```

6563 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6564 {
6565     \int_set:Nn \l_tmpa_int { #1 }
6566     \int_set:Nn \l_tmpb_int { #2 }
6567     \bool_set_false:N \l_tmpb_bool
6568     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6569         { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6570 }

```

```

6571 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6572 {
6573   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6574   {
6575     \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6576     {
6577       \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6578       {
6579         \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6580         { \bool_set_true:N \l_tmpb_bool }
6581       }
6582     }
6583   }
6584 }

```

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.

```

6585 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6586 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6587 {
6588   auto-columns-width .code:n =
6589   {
6590     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6591     \dim_gzero_new:N \g_@@_max_cell_width_dim
6592     \bool_set_true:N \l_@@_auto_columns_width_bool
6593   }
6594 }

6595 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6596 {
6597   \int_gincr:N \g_@@_NiceMatrixBlock_int
6598   \dim_zero:N \l_@@_columns_width_dim
6599   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6600   \bool_if:NT \l_@@_block_auto_columns_width_bool
6601   {
6602     \cs_if_exist:cT
6603     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6604     {
6605       % is \exp_args:NNe mandatory?
6606       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6607       {
6608         \use:c
6609         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6610       }
6611     }
6612   }
6613 }

```

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).

```

6614 {
6615   \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.

```

6616 { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6617 {
6618   \bool_if:NT \l_@@_block_auto_columns_width_bool
6619   {
6620     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6621     \iow_shipout:Nx \@mainaux
6622     {
6623       \cs_gset:cpn
6624       { @@ _ 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.

```

6625       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6626     }
6627     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6628   }
6629 }
6630 \ignorespacesafterend
6631 }

```

26 The extra nodes

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

```

6632 \cs_generate_variant:Nn \dim_min:nn { v n }
6633 \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).

```

6634 \cs_new_protected:Npn \@@_create_extra_nodes:
6635 {
6636   \bool_if:nTF \l_@@_medium_nodes_bool
6637   {
6638     \bool_if:nTF \l_@@_large_nodes_bool
6639     \@@_create_medium_and_large_nodes:
6640     \@@_create_medium_nodes:
6641   }
6642   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6643 }

```

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`.

```

6644 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6645 {
6646   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6647   {
6648     \dim_zero_new:c { \l_@@_row_\@@_i: _min_dim }
6649     \dim_set_eq:cN { \l_@@_row_\@@_i: _min_dim } \c_max_dim
6650     \dim_zero_new:c { \l_@@_row_\@@_i: _max_dim }
6651     \dim_set:cn { \l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6652   }
6653   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6654   {
6655     \dim_zero_new:c { \l_@@_column_\@@_j: _min_dim }
6656     \dim_set_eq:cN { \l_@@_column_\@@_j: _min_dim } \c_max_dim
6657     \dim_zero_new:c { \l_@@_column_\@@_j: _max_dim }
6658     \dim_set:cn { \l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6659   }

```

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

```

6660   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6661   {
6662     \int_step_variable:nnNn
6663     \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.

```

6664     {
6665       \cs_if_exist:cT
6666       { \pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

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

```

6667     {
6668       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6669       \dim_set:cn { \l_@@_row_\@@_i: _min_dim }
6670       { \dim_min:vn { \l_@@_row _ \@@_i: _min_dim } \pgf@y }
6671       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6672       {
6673         \dim_set:cn { \l_@@_column _ \@@_j: _min_dim }
6674         { \dim_min:vn { \l_@@_column _ \@@_j: _min_dim } \pgf@x }
6675       }

```

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

```

6676       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6677       \dim_set:cn { \l_@@_row _ \@@_i: _ max_dim }
6678       { \dim_max:vn { \l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6679       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6680       {
6681         \dim_set:cn { \l_@@_column _ \@@_j: _ max_dim }
6682         { \dim_max:vn { \l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6683       }
6684     }
6685   }
6686 }

```

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

```

6687   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6688   {
6689     \dim_compare:nNnT
6690     { \dim_use:c { \l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6691     {
6692       \@@_qpoint:n { row - \@@_i: - base }

```

```

6693         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6694         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6695     }
6696 }
6697 \int_step_variable:nNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6698 {
6699     \dim_compare:nNnT
6700     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6701     {
6702         \@@_qpoint:n { col - \@@_j: }
6703         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6704         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6705     }
6706 }
6707 }

```

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

```

6708 \cs_new_protected:Npn \@@_create_medium_nodes:
6709 {
6710     \pgfpicture
6711     \pgfrememberpicturepositiononpagetrue
6712     \pgf@relevantforpicturesizefalse
6713     \@@_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”.

```

6714         \tl_set:Nn \l_@@_suffix_tl { -medium }
6715         \@@_create_nodes:
6716     \endpgfpicture
6717 }

```

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:`.

```

6718 \cs_new_protected:Npn \@@_create_large_nodes:
6719 {
6720     \pgfpicture
6721     \pgfrememberpicturepositiononpagetrue
6722     \pgf@relevantforpicturesizefalse
6723     \@@_computations_for_medium_nodes:
6724     \@@_computations_for_large_nodes:
6725     \tl_set:Nn \l_@@_suffix_tl { - large }
6726     \@@_create_nodes:
6727     \endpgfpicture
6728 }
6729 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6730 {
6731     \pgfpicture
6732     \pgfrememberpicturepositiononpagetrue
6733     \pgf@relevantforpicturesizefalse
6734     \@@_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”.

```

6735         \tl_set:Nn \l_@@_suffix_tl { - medium }
6736         \@@_create_nodes:

```

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

```

6737     \@@_computations_for_large_nodes:
6738     \tl_set:Nn \l_@@_suffix_tl { - large }
6739     \@@_create_nodes:
6740     \endpgfpicture
6741 }

```

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.

```

6742 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6743 {
6744     \int_set:Nn \l_@@_first_row_int 1
6745     \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`.

```

6746     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6747     {
6748         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6749         {
6750             (
6751                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6752                 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6753             )
6754             / 2
6755         }
6756         \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6757         { l_@@_row _ \@@_i: _ min_dim }
6758     }
6759     \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6760     {
6761         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6762         {
6763             (
6764                 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6765                 \dim_use:c
6766                 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6767             )
6768             / 2
6769         }
6770         \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6771         { l_@@_column _ \@@_j: _ max _ dim }
6772     }

```

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

```

6773     \dim_sub:cn
6774     { l_@@_column _ 1 _ min _ dim }
6775     \l_@@_left_margin_dim
6776     \dim_add:cn
6777     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6778     \l_@@_right_margin_dim
6779 }

```

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).

```

6780 \cs_new_protected:Npn \@@_create_nodes:
6781 {
6782     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6783     {

```

```

6784 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6785 {

```

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

```

6786 \@@_pgf_rect_node:nnnnn
6787 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6788 { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6789 { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6790 { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6791 { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6792 \str_if_empty:NF \l_@@_name_str
6793 {
6794 \pgfnodealias
6795 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6796 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6797 }
6798 }
6799 }

```

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 .

```

6800 \seq_map_pairwise_function:NNN
6801 \g_@@_multicolumn_cells_seq
6802 \g_@@_multicolumn_sizes_seq
6803 \@@_node_for_multicolumn:nn
6804 }

```

```

6805 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6806 {
6807 \cs_set_nopar:Npn \@@_i: { #1 }
6808 \cs_set_nopar:Npn \@@_j: { #2 }
6809 }

```

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”).

```

6810 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6811 {
6812 \@@_extract_coords_values: #1 \q_stop
6813 \@@_pgf_rect_node:nnnnn
6814 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6815 { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6816 { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6817 { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
6818 { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6819 \str_if_empty:NF \l_@@_name_str
6820 {
6821 \pgfnodealias
6822 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6823 { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6824 }
6825 }

```

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.

```

6826 \keys_define:nn { NiceMatrix / Block / FirstPass }
6827 {
6828   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6829   l .value_forbidden:n = true ,
6830   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6831   r .value_forbidden:n = true ,
6832   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6833   c .value_forbidden:n = true ,
6834   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6835   L .value_forbidden:n = true ,
6836   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6837   R .value_forbidden:n = true ,
6838   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6839   C .value_forbidden:n = true ,
6840   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6841   t .value_forbidden:n = true ,
6842   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6843   T .value_forbidden:n = true ,
6844   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6845   b .value_forbidden:n = true ,
6846   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6847   B .value_forbidden:n = true ,
6848   color .code:n =
6849     \@@_color:n { #1 }
6850     \tl_set_rescan:Nnn
6851       \l_@@_draw_tl
6852       { \char_set_catcode_other:N ! }
6853       { #1 } ,
6854   color .value_required:n = true ,
6855   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6856   respect-arraystretch .default:n = true
6857 }

```

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.

```

6858 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

6859 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6860 {

```

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).

```

6861   \peek_remove_spaces:n
6862   {
6863     \tl_if_blank:nTF { #2 }
6864     { \@@_Block_i 1-1 \q_stop }
6865     {
6866       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6867       \@@_Block_i_czech \@@_Block_i
6868       #2 \q_stop
6869     }
6870     { #1 } { #3 } { #4 }
6871   }
6872 }

```

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

```

6873 \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`.

```

6874 {
6875   \char_set_catcode_active:N -
6876   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6877 }

```

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.

```

6878 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6879 {

```

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).

```

6880   \bool_lazy_or:nnTF
6881     { \tl_if_blank_p:n { #1 } }
6882     { \str_if_eq_p:nn { #1 } { * } }
6883     { \int_set:Nn \l_tmpa_int { 100 } }
6884     { \int_set:Nn \l_tmpa_int { #1 } }
6885   \bool_lazy_or:nnTF
6886     { \tl_if_blank_p:n { #2 } }
6887     { \str_if_eq_p:nn { #2 } { * } }
6888     { \int_set:Nn \l_tmpb_int { 100 } }
6889     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6890   \int_compare:nNnTF \l_tmpb_int = 1
6891   {
6892     \str_if_empty:NTF \l_@@_hpos_cell_str
6893     { \str_set:Nn \l_@@_hpos_block_str c }
6894     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6895   }
6896   { \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.

```

6897   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6898   \tl_set:Nx \l_tmpa_tl
6899   {
6900     { \int_use:N \c@iRow }
6901     { \int_use:N \c@jCol }
6902     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6903     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6904   }

```

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).

```

6905   \bool_if:nTF
6906   {
6907     (
6908       \int_compare_p:nNn { \l_tmpa_int } = 1
6909       ||

```

```

6910     \int_compare:nNn { \l_tmpb_int } = 1
6911   )
6912   && ! \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.

```

6913     && ! \l_@@_X_column_bool
6914   }
6915   { \exp_args:Nee \@@_Block_iv:nnnnn }
6916   { \exp_args:Nee \@@_Block_v:nnnnn }
6917   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6918 }

```

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 math mode and before the composition of the block and #5 is the label (=content) of the block.

```

6919 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6920 {
6921   \int_gincr:N \g_@@_block_box_int
6922   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6923   {
6924     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6925     {
6926       \@@_actually_diagbox:nnnnnn
6927       { \int_use:N \c@iRow }
6928       { \int_use:N \c@jCol }
6929       { \int_eval:n { \c@iRow + #1 - 1 } }
6930       { \int_eval:n { \c@jCol + #2 - 1 } }
6931       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6932     }
6933   }
6934   \box_gclear_new:c
6935   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the \Block in a TeX box. *Be careful:* if after, the construction of the box, the boolean \g_@@_rotate_bool is raised (which means that the command \rotate was present in the content of the \Block) we will rotate the box but also, maybe, change the position of the baseline!

```

6936   \hbox_gset:cn
6937   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
6938   {

```

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}).

```

6939   \tl_if_empty:NTF \l_@@_color_tl
6940   { \int_compare:nNnT { #2 } = 1 \set@color }
6941   { \@@_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.

```

6942   \int_compare:nNnT { #1 } = 1

```

```

6943     {
6944         \int_if_zero:nTF \c@iRow
6945         \l_@@_code_for_first_row_tl
6946         {
6947             \int_compare:nNtT \c@iRow = \l_@@_last_row_int
6948             \l_@@_code_for_last_row_tl
6949         }
6950         \g_@@_row_style_tl
6951     }
6952     \bool_if:NF \l_@@_respect_arraystretch_bool
6953     { \cs_set:Npn \arraystretch { 1 } }
6954     \dim_zero:N \extrarowheight

```

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

```

6955     #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.).

```

6956     \@@_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}`.

```

6957     \bool_if:NTF \l_@@_tabular_bool
6958     {
6959         \bool_lazy_all:nTF
6960         {
6961             { \int_compare_p:nNn { #2 } = 1 }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventionnal value of -1 cm.

```

6962         { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6963         { ! \g_@@_rotate_bool }
6964     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

6965     {
6966         \use:e
6967         {
6968             \exp_not:N \begin { minipage }%
6969             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6970             { \l_@@_col_width_dim }
6971             \str_case:Vn \l_@@_hpos_block_str
6972             { c \centering r \raggedleft l \raggedright }
6973         }
6974         #5
6975     \end { minipage }
6976 }

```

In the other cases, we use a `{tabular}`.

```

6977     {
6978         \use:e
6979         {
6980             \exp_not:N \begin { tabular }%
6981             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6982             { @ { } \l_@@_hpos_block_str @ { } }
6983         }
6984         #5
6985     \end { tabular }
6986 }
6987 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

6988     {
6989         \c_math_toggle_token
6990         \use:e
6991         {
6992             \exp_not:N \begin { array }%
6993             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6994             { @ { } \l_@@_hpos_block_str @ { } }
6995         }
6996         #5
6997         \end { array }
6998         \c_math_toggle_token
6999     }
7000 }

```

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).

```

7001 \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.

```

7002 \int_compare:nNnT { #2 } = 1
7003 {
7004     \dim_gset:Nn \g_@@_blocks_wd_dim
7005     {
7006         \dim_max:nn
7007         \g_@@_blocks_wd_dim
7008         {
7009             \box_wd:c
7010             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7011         }
7012     }
7013 }

```

If we are in a mono-row block, we take into account the height and the depth of that block for the height and the depth of the row.

```

7014 \int_compare:nNnT { #1 } = 1
7015 {
7016     \dim_gset:Nn \g_@@_blocks_ht_dim
7017     {
7018         \dim_max:nn
7019         \g_@@_blocks_ht_dim
7020         {
7021             \box_ht:c
7022             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7023         }
7024     }
7025     \dim_gset:Nn \g_@@_blocks_dp_dim
7026     {
7027         \dim_max:nn
7028         \g_@@_blocks_dp_dim
7029         {
7030             \box_dp:c
7031             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7032         }
7033     }
7034 }
7035 \seq_gput_right:Nx \g_@@_blocks_seq
7036 {
7037     \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.

```

7038     {
7039         \exp_not:n { #3 } ,
7040         \l_@@_hpos_block_str ,
Now, we put a key for the vertical alignment.
7041         \bool_if:NT \g_@@_rotate_bool
7042         {
7043             \bool_if:NTF \g_@@_rotate_c_bool
7044             { v-center }
7045             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7046         }
7047     }
7048 }
7049 {
7050     \box_use_drop:c
7051     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7052 }
7053 }
7054 \bool_set_false:N \g_@@_rotate_c_bool
7055 }

7056 \cs_new:Npn \@@_adjust_hpos_rotate:
7057 {
7058     \bool_if:NT \g_@@_rotate_bool
7059     {
7060         \str_set:Nx \l_@@_hpos_block_str
7061         {
7062             \bool_if:NTF \g_@@_rotate_c_bool
7063             { c }
7064             {
7065                 \str_case:VnF \l_@@_vpos_of_block_str
7066                 { b l B l t r T r }
7067                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7068             }
7069         }
7070     }
7071 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustement of the baseline of the block*.

```

7072 \cs_new_protected:Npn \@@_rotate_box_of_block:
7073 {
7074     \box_grotate:cn
7075     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7076     { 90 }
7077     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7078     {
7079         \vbox_gset_top:cn
7080         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7081         {
7082             \skip_vertical:n { 0.8 ex }
7083             \box_use:c
7084             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7085         }
7086     }
7087     \bool_if:NT \g_@@_rotate_c_bool
7088     {
7089         \hbox_gset:cn

```

```

7090 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7091 {
7092   \c_math_toggle_token
7093   \vcenter
7094   {
7095     \box_use:c
7096     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7097   }
7098   \c_math_toggle_token
7099 }
7100 }
7101 }

```

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.

```

7102 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7103 {
7104   \seq_gput_right:Nx \g_@@_blocks_seq
7105   {
7106     \l_tmpa_tl
7107     { \exp_not:n { #3 } }
7108     {
7109       \bool_if:NTF \l_@@_tabular_bool
7110       {
7111         \group_begin:
7112         \bool_if:NF \l_@@_respect_arraystretch_bool
7113         { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7114         \exp_not:n
7115         {
7116           \dim_zero:N \extrarowheight
7117           #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).

```

7118       % \@@_adjust_hpos_rotate:
7119       \use:e
7120       {
7121         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
7122         { @ { } \l_@@_hpos_block_str @ { } }
7123       }
7124       #5
7125       \end { tabular }
7126     }
7127   \group_end:
7128 }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7129 {
7130   \group_begin:
7131   \bool_if:NF \l_@@_respect_arraystretch_bool
7132   { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7133   \exp_not:n
7134   {
7135     \dim_zero:N \extrarowheight
7136     #4

```

```

7137         % \@@_adjust_hpos_rotate:
7138         \c_math_toggle_token    % :n c
7139         \use:e
7140         {
7141             \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
7142             { @ { } \l_@@_hpos_block_str @ { } }
7143         }
7144         #5
7145         \end { array }
7146         \c_math_toggle_token
7147     }
7148     \group_end:
7149 }
7150 }
7151 }
7152 }

```

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).

```

7153 \keys_define:nn { NiceMatrix / Block / SecondPass }
7154 {
7155     tikz .code:n =
7156         \IfPackageLoadedTF { tikz }
7157         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7158         { \@@_error:n { tikz-key-without~tikz } } ,
7159     tikz .value_required:n = true ,
7160     fill .code:n =
7161         \tl_set_rescan:Nnn
7162         \l_@@_fill_tl
7163         { \char_set_catcode_other:N ! }
7164         { #1 } ,
7165     fill .value_required:n = true ,
7166     opacity .tl_set:N = \l_@@_opacity_tl ,
7167     opacity .value_required:n = true ,
7168     draw .code:n =
7169         \tl_set_rescan:Nnn
7170         \l_@@_draw_tl
7171         { \char_set_catcode_other:N ! }
7172         { #1 } ,
7173     draw .default:n = default ,
7174     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7175     rounded-corners .default:n = 4 pt ,
7176     color .code:n =
7177         \@@_color:n { #1 }
7178         \tl_set_rescan:Nnn
7179         \l_@@_draw_tl
7180         { \char_set_catcode_other:N ! }
7181         { #1 } ,
7182     borders .clist_set:N = \l_@@_borders_clist ,
7183     borders .value_required:n = true ,
7184     hvlines .meta:n = { vlines , hlines } ,
7185     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7186     vlines .default:n = true ,
7187     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7188     hlines .default:n = true ,
7189     line-width .dim_set:N = \l_@@_line_width_dim ,
7190     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7191     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7192     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,

```

```

7193   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7194   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7195           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7196   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7197           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7198   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7199           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7200   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
7201   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
7202   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
7203   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
7204   v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
7205   v-center .value_forbidden:n = true ,
7206   name .tl_set:N = \l_@@_block_name_str ,
7207   name .value_required:n = true ,
7208   name .initial:n = ,
7209   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
7210   transparent .bool_set:N = \l_@@_transparent_bool ,
7211   transparent .default:n = true ,
7212   transparent .initial:n = false ,
7213   unknown .code:n = @@_error:n { Unknown~key~for~Block }
7214 }

```

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.

```

7215 \cs_new_protected:Npn \@@_draw_blocks:
7216 {
7217   \cs_set_eq:NN \ialign \@@_old_ialign:
7218   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7219 }
7220 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7221 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7222   \int_zero_new:N \l_@@_last_row_int
7223   \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.

```

7224   \int_compare:nNnTF { #3 } > { 99 }
7225     { \int_set_eq:NN \l_@@_last_row_int \c{iRow} }
7226     { \int_set:Nn \l_@@_last_row_int { #3 } }
7227   \int_compare:nNnTF { #4 } > { 99 }
7228     { \int_set_eq:NN \l_@@_last_col_int \c{jCol} }
7229     { \int_set:Nn \l_@@_last_col_int { #4 } }
7230   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7231     {
7232       \bool_lazy_and:nnTF
7233         \l_@@_preamble_bool
7234         {
7235           \int_compare_p:n
7236             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7237         }
7238     {
7239       \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
7240       \@@_msg_redirect_name:nn { Block~too~large~2 } { none }

```



```

7241         \@@_msg_redirect_name:nn { columns~not~used } { none }
7242     }
7243     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7244 }
7245 {
7246     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7247     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7248     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7249 }
7250 }

```

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

```

7251 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7252 {

```

The group is for the keys.

```

7253     \group_begin:
7254     \int_compare:nNnT { #1 } = { #3 }
7255     { \str_set:Nn \l_@@_vpos_of_block_str { t } }
7256     \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7257     \bool_if:NT \l_@@_vlines_block_bool
7258     {
7259         \tl_gput_right:Nx \g_nicematrix_code_after_tl
7260         {
7261             \@@_vlines_block:nnn
7262             { \exp_not:n { #5 } }
7263             { #1 - #2 }
7264             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7265         }
7266     }
7267     \bool_if:NT \l_@@_hlines_block_bool
7268     {
7269         \tl_gput_right:Nx \g_nicematrix_code_after_tl
7270         {
7271             \@@_hlines_block:nnn
7272             { \exp_not:n { #5 } }
7273             { #1 - #2 }
7274             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7275         }
7276     }
7277     \bool_if:nF
7278     {
7279         \l_@@_transparent_bool
7280         || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7281     }
7282     {

```

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).

```

7283         \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7284         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7285     }

7286     \bool_lazy_and:nnT
7287     { ! ( \tl_if_empty_p:N \l_@@_draw_tl ) }
7288     { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7289     { \@@_error:n { hlines~with~color } }

7290     \tl_if_empty:NF \l_@@_draw_tl
7291     {

```

```

7292 \tl_gput_right:Nx \g_nicematrix_code_after_tl
7293 {
7294   \@@_stroke_block:nnn
7295   { \exp_not:n { #5 } } % #5 are the options
7296   { #1 - #2 }
7297   { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7298 }
7299 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7300 { { #1 } { #2 } { #3 } { #4 } }
7301 }
7302 \clist_if_empty:NF \l_@@_borders_clist
7303 {
7304   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7305   {
7306     \@@_stroke_borders_block:nnn
7307     { \exp_not:n { #5 } }
7308     { #1 - #2 }
7309     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7310   }
7311 }
7312 \tl_if_empty:NF \l_@@_fill_tl
7313 {
7314   \tl_if_empty:NF \l_@@_opacity_tl
7315   {
7316     \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7317       {
7318         \tl_set:Nx \l_@@_fill_tl
7319         {
7320           [ opacity = \l_@@_opacity_tl ,
7321           \tl_tail:V \l_@@_fill_tl
7322         }
7323       }
7324       {
7325         \tl_set:Nx \l_@@_fill_tl
7326         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7327       }
7328     }
7329     \tl_gput_right:Nx \g_@@_pre_code_before_tl
7330     {
7331       \exp_not:N \roundedrectanglecolor
7332       \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7333         { \l_@@_fill_tl }
7334         { { \l_@@_fill_tl } }
7335         { #1 - #2 }
7336         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7337         { \dim_use:N \l_@@_rounded_corners_dim }
7338       }
7339     }
7340     \seq_if_empty:NF \l_@@_tikz_seq
7341     {
7342       \tl_gput_right:Nx \g_nicematrix_code_before_tl
7343       {
7344         \@@_block_tikz:nnnnn
7345         { #1 }
7346         { #2 }
7347         { \int_use:N \l_@@_last_row_int }
7348         { \int_use:N \l_@@_last_col_int }
7349         { \seq_use:Nn \l_@@_tikz_seq { , } }
7350       }
7351     }
7352     \cs_set_protected_nopar:Npn \diagbox ##1 ##2

```

```

7353 {
7354   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7355   {
7356     \@@_actually_diagbox:nnnnnn
7357     { #1 }
7358     { #2 }
7359     { \int_use:N \l_@@_last_row_int }
7360     { \int_use:N \l_@@_last_col_int }
7361     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7362   }
7363 }

7364 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7365 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one      & \\\
                        &      & two      & \\\
three                  & four & five      & \\\
six                    & seven & eight     & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

7366 \pgfpicture
7367 \pgfrememberpicturepositiononpagetrue
7368 \pgf@relevantforpicturesizefalse
7369 \@@_qpoint:n { row - #1 }
7370 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7371 \@@_qpoint:n { col - #2 }
7372 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7373 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7374 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7375 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7376 \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.

```

7377 \@@_pgf_rect_node:nnnnn
7378 { \@@_env: - #1 - #2 - block }
7379 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7380 \str_if_empty:NF \l_@@_block_name_str
7381 {
7382   \pgfnodealias
7383   { \@@_env: - \l_@@_block_name_str }
7384   { \@@_env: - #1 - #2 - block }
7385   \str_if_empty:NF \l_@@_name_str
7386   {
7387     \pgfnodealias
7388     { \l_@@_name_str - \l_@@_block_name_str }

```

```

7389         { \@@_env: - #1 - #2 - block }
7390     }
7391 }

```

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.

```

7392 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7393 {
7394     \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.

```

7395     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7396     {

```

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.

```

7397         \cs_if_exist:cT
7398         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7399         {
7400             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7401             {
7402                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7403                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7404             }
7405         }
7406     }

```

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.

```

7407     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7408     {
7409         \@@_qpoint:n { col - #2 }
7410         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7411     }
7412 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7413 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7414 {
7415     \cs_if_exist:cT
7416     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7417     {
7418         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7419         {
7420             \pgfpointanchor
7421             { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7422             { east }
7423             \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7424         }
7425     }
7426 }
7427 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7428 {
7429     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7430     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7431 }
7432 \@@_pgf_rect_node:nnnnn
7433 { \@@_env: - #1 - #2 - block - short }
7434 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7435 }

```

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.

```

7436 \bool_if:NT \l_@@_medium_nodes_bool
7437 {
7438   \@@_pgf_rect_node:nnn
7439   { \@@_env: - #1 - #2 - block - medium }
7440   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7441   {
7442     \pgfpointanchor
7443     { \@@_env:
7444       - \int_use:N \l_@@_last_row_int
7445       - \int_use:N \l_@@_last_col_int - medium
7446     }
7447     { south-east }
7448   }
7449 }

```

Now, we will put the label of the block.

```

7450 \bool_lazy_any:nTF
7451 {
7452   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7453   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7454   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7455 }

```

```

7456 {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7457   \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.

```

7458   \bool_if:nT \g_@@_last_col_found_bool
7459   {
7460     \int_compare:nNnT { #2 } = \g_@@_col_total_int
7461     { \str_set:Nn \l_@@_hpos_block_str l }
7462   }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7463   \tl_set:Nx \l_tmpa_tl
7464   {
7465     \str_case:Vn \l_@@_vpos_of_block_str
7466     {
7467       c {
7468         \str_case:Vn \l_@@_hpos_block_str
7469         {
7470           c { center }
7471           l { west }
7472           r { east }
7473         }
7474       }
7475     }
7476     T {
7477       \str_case:Vn \l_@@_hpos_block_str
7478       {
7479         c { north }
7480         l { north-west }
7481         r { north-east }
7482       }
7483     }
7484     B {
7485       \str_case:Vn \l_@@_hpos_block_str
7486       {
7487         c { south }
7488         l { south-west }
7489         r { south-east }
7490       }

```

```

7491         }
7492     }
7493 }
7494 }
7495 }
7496 \pgftransformshift
7497 {
7498     \pgfpointanchor
7499     {
7500         \@@_env: - #1 - #2 - block
7501         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7502     }
7503     { \l_tmpa_tl }
7504 }
7505 \pgfset
7506 {
7507     inner~xsep = \c_zero_dim ,
7508     inner~ysep = \l_@@_block_ysep_dim
7509 }
7510 \pgfnode
7511 { rectangle }
7512 { \l_tmpa_tl }
7513 { \box_use_drop:N \l_@@_cell_box } { } { }
7514 }

```

End of the case when $\l_@@_vpos_of_block_str$ is equal to c, T or B. Now, the other cases.

```

7515 {
7516     \pgfextracty \l_tmpa_dim
7517     {
7518         \@@_qpoint:n
7519         {
7520             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7521             - base
7522         }
7523     }
7524     \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.

```

7525 \pgfpointanchor
7526 {
7527     \@@_env: - #1 - #2 - block
7528     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7529 }
7530 {
7531     \str_case:Vn \l_@@_hpos_block_str
7532     {
7533         c { center }
7534         l { west }
7535         r { east }
7536     }
7537 }

```

We put the label of the block which has been composed in $\l_@@_cell_box$.

```

7538 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7539 \pgfset { inner~sep = \c_zero_dim }
7540 \pgfnode
7541 { rectangle }
7542 {
7543     \str_case:Vn \l_@@_hpos_block_str
7544     {
7545         c { base }
7546         l { base~west }
7547         r { base~east }
7548     }

```

```

7549     }
7550     { \box_use_drop:N \l_@@_cell_box } { } { }
7551   }
7552   \endpgfpicture
7553   \group_end:
7554 }

```

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).

```

7555 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7556 {
7557   \group_begin:
7558   \tl_clear:N \l_@@_draw_tl
7559   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7560   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7561   \pgfpicture
7562   \pgfrememberpicturepositiononpagetrue
7563   \pgf@relevantforpicturesizefalse
7564   \tl_if_empty:NF \l_@@_draw_tl
7565   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7566     \str_if_eq:VnTF \l_@@_draw_tl { default }
7567     { \CT@arc@ }
7568     { \@@_color:V \l_@@_draw_tl }
7569   }
7570   \pgfsetcornersarced
7571   {
7572     \pgfpoint
7573     { \l_@@_rounded_corners_dim }
7574     { \l_@@_rounded_corners_dim }
7575   }
7576   \@@_cut_on_hyphen:w #2 \q_stop
7577   \bool_lazy_and:nnT
7578   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7579   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7580   {
7581     \@@_qpoint:n { row - \l_tmpa_tl }
7582     \dim_set_eq:NN \l_tmpb_dim \pgf@y
7583     \@@_qpoint:n { col - \l_tmpb_tl }
7584     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7585     \@@_cut_on_hyphen:w #3 \q_stop
7586     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7587     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7588     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7589     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7590     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7591     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7592     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7593     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7594     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7595     \pgfpathrectanglecorners
7596     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7597     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7598     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7599     { \pgfusepathqstroke }
7600     { \pgfusepath { stroke } }
7601   }
7602   \endpgfpicture
7603   \group_end:
7604 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7605 \keys_define:nn { NiceMatrix / BlockStroke }
7606 {
7607   color .tl_set:N = \l_@@_draw_tl ,
7608   draw .code:n =
7609     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7610   draw .default:n = default ,
7611   line-width .dim_set:N = \l_@@_line_width_dim ,
7612   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7613   rounded-corners .default:n = 4 pt
7614 }

```

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).

```

7615 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7616 {
7617   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7618   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7619   \@@_cut_on_hyphen:w #2 \q_stop
7620   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7621   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7622   \@@_cut_on_hyphen:w #3 \q_stop
7623   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7624   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7625   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7626   {
7627     \use:e
7628     {
7629       \@@_vline:n
7630       {
7631         position = ##1 ,
7632         start = \l_@@_tmpc_tl ,
7633         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7634         total-width = \dim_use:N \l_@@_line_width_dim
7635       }
7636     }
7637   }
7638 }
7639 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7640 {
7641   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7642   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7643   \@@_cut_on_hyphen:w #2 \q_stop
7644   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7645   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7646   \@@_cut_on_hyphen:w #3 \q_stop
7647   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7648   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7649   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7650   {
7651     \use:e
7652     {
7653       \@@_hline:n
7654       {
7655         position = ##1 ,
7656         start = \l_@@_tmpd_tl ,
7657         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7658         total-width = \dim_use:N \l_@@_line_width_dim
7659       }
7660     }
7661   }
7662 }

```


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).

```

7663 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7664 {
7665   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7666   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7667   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7668     { \@@_error:n { borders~forbidden } }
7669     {
7670       \tl_clear_new:N \l_@@_borders_tikz_tl
7671       \keys_set:nV
7672         { NiceMatrix / OnlyForTikzInBorders }
7673         \l_@@_borders_clist
7674       \@@_cut_on_hyphen:w #2 \q_stop
7675       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7676       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7677       \@@_cut_on_hyphen:w #3 \q_stop
7678       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7679       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7680       \@@_stroke_borders_block_i:
7681     }
7682   }
7683 \hook_gput_code:nnn { begindocument } { . }
7684 {
7685   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7686   {
7687     \c_@@_pgfortikzpicture_tl
7688     \@@_stroke_borders_block_ii:
7689     \c_@@_endpgfortikzpicture_tl
7690   }
7691 }
7692 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7693 {
7694   \pgfrememberpicturepositiononpagetrue
7695   \pgf@relevantforpicturesizefalse
7696   \CT@arc@
7697   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7698   \clist_if_in:NnT \l_@@_borders_clist { right }
7699     { \@@_stroke_vertical:n \l_tmpb_tl }
7700   \clist_if_in:NnT \l_@@_borders_clist { left }
7701     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7702   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7703     { \@@_stroke_horizontal:n \l_tmpa_tl }
7704   \clist_if_in:NnT \l_@@_borders_clist { top }
7705     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7706 }
7707 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7708 {
7709   tikz .code:n =
7710     \cs_if_exist:NTF \tikzpicture
7711       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7712       { \@@_error:n { tikz~in~borders~without~tikz } } ,
7713   tikz .value_required:n = true ,
7714   top .code:n = ,
7715   bottom .code:n = ,
7716   left .code:n = ,
7717   right .code:n = ,
7718   unknown .code:n = \@@_error:n { bad~border }
7719 }

```

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).

```

7720 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7721 {
7722   \@@_qpoint:n \l_@@_tmpc_tl
7723   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7724   \@@_qpoint:n \l_tmpa_tl
7725   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7726   \@@_qpoint:n { #1 }
7727   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7728   {
7729     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7730     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7731     \pgfusepathqstroke
7732   }
7733   {
7734     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7735     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7736   }
7737 }

```

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).

```

7738 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7739 {
7740   \@@_qpoint:n \l_@@_tmpd_tl
7741   \clist_if_in:NnTF \l_@@_borders_clist { left }
7742   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7743   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7744   \@@_qpoint:n \l_tmpb_tl
7745   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7746   \@@_qpoint:n { #1 }
7747   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7748   {
7749     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7750     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7751     \pgfusepathqstroke
7752   }
7753   {
7754     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7755     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7756   }
7757 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7758 \keys_define:nn { NiceMatrix / BlockBorders }
7759 {
7760   borders .clist_set:N = \l_@@_borders_clist ,
7761   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7762   rounded-corners .default:n = 4 pt ,
7763   line-width .dim_set:N = \l_@@_line_width_dim
7764 }

```

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.

```

7765 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7766 {
7767   \begin { tikzpicture }
7768   \@@_clip_with_rounded_corners:
7769   \clist_map_inline:nn { #5 }

```

```

7770 {
7771   \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7772   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7773   (
7774     [
7775       xshift = \dim_use:N \l_@@_offset_dim ,
7776       yshift = - \dim_use:N \l_@@_offset_dim
7777     ]
7778     #1 -| #2
7779   )
7780   rectangle
7781   (
7782     [
7783       xshift = - \dim_use:N \l_@@_offset_dim ,
7784       yshift = \dim_use:N \l_@@_offset_dim
7785     ]
7786     \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7787   ) ;
7788 }
7789 \end { tikzpicture }
7790 }
7791 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7792 \keys_define:nn { NiceMatrix / SpecialOffset }
7793 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7794 \cs_set_protected:Npn \@@_renew_matrix:
7795 {
7796   \RenewDocumentEnvironment { pmatrix } { } {
7797     { \pNiceMatrix }
7798     { \endpNiceMatrix }
7799   }
7800   \RenewDocumentEnvironment { vmatrix } { } {
7801     { \vNiceMatrix }
7802     { \endvNiceMatrix }
7803   }
7804   \RenewDocumentEnvironment { Vmatrix } { } {
7805     { \VNiceMatrix }
7806     { \endVNiceMatrix }
7807   }
7808   \RenewDocumentEnvironment { bmatrix } { } {
7809     { \bNiceMatrix }
7810     { \endbNiceMatrix }
7811   }
7812   \RenewDocumentEnvironment { Bmatrix } { } {
7813     { \BNiceMatrix }
7814     { \endBNiceMatrix }
7815   }
7816 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7812 \keys_define:nn { NiceMatrix / Auto }
7813 {
7814   columns-type .tl_set:N = \l_@@_columns_type_tl ,
7815   columns-type .value_required:n = true ,
7816   l .meta:n = { columns-type = l } ,
7817   r .meta:n = { columns-type = r } ,
7818   c .meta:n = { columns-type = c } ,
7819   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

7820 delimiters / color .value_required:n = true ,
7821 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7822 delimiters / max-width .default:n = true ,
7823 delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7824 delimiters .value_required:n = true ,
7825 rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7826 rounded-corners .default:n = 4 pt
7827 }

7828 \NewDocumentCommand \AutoNiceMatrixWithDelims
7829 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7830 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

7831 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7832 {

```

The group is for the protection of the keys.

```

7833 \group_begin:
7834 \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`).

```

7835 % \bool_set_false:N \l_@@_preamble_bool
7836 \use:e
7837 {
7838   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7839   { * { #4 } { \exp_not:N \l_@@_columns_type_tl } }
7840   [ \exp_not:N \l_tmpa_tl ]
7841 }
7842 \int_if_zero:nT \l_@@_first_row_int
7843 {
7844   \int_if_zero:nT \l_@@_first_col_int { & }
7845   \prg_replicate:nn { #4 - 1 } { & }
7846   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7847 }
7848 \prg_replicate:nn { #3 }
7849 {
7850   \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`).

```

7851   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7852   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7853 }
7854 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7855 {
7856   \int_if_zero:nT \l_@@_first_col_int { & }
7857   \prg_replicate:nn { #4 - 1 } { & }
7858   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7859 }
7860 \end { NiceArrayWithDelims }
7861 \group_end:
7862 }

7863 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7864 {
7865   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7866   {
7867     \bool_gset_true:N \g_@@_delims_bool
7868     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7869     \AutoNiceMatrixWithDelims { #2 } { #3 }
7870   }
7871 }

```

```

7872 \@@_define_com:nnn p ( )
7873 \@@_define_com:nnn b [ ]
7874 \@@_define_com:nnn v | |
7875 \@@_define_com:nnn V \| \|
7876 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7877 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
7878 {
7879   \group_begin:
7880   \bool_gset_false:N \g_@@_delims_bool
7881   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7882   \group_end:
7883 }

```

30 The redefinition of the command `\dotfill`

```

7884 \cs_set_eq:NN \@@_old_dotfill \dotfill
7885 \cs_new_protected:Npn \@@_dotfill:
7886 {

```

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}`).

```

7887   \@@_old_dotfill
7888   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7889 }

```

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`.

```

7890 \cs_new_protected:Npn \@@_dotfill_i:
7891 { \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.

```

7892 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7893 {
7894   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7895   {
7896     \@@_actually_diagbox:nnnnnn
7897     { \int_use:N \c@iRow }
7898     { \int_use:N \c@jCol }
7899     { \int_use:N \c@iRow }
7900     { \int_use:N \c@jCol }
7901     { \exp_not:n { #1 } }
7902     { \exp_not:n { #2 } }
7903   }

```

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`.

```

7904   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7905   {
7906     { \int_use:N \c@iRow }
7907     { \int_use:N \c@jCol }
7908     { \int_use:N \c@iRow }
7909     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7910         { }
7911     }
7912 }

```

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.

```

7913 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7914 {
7915     \pgfpicture
7916     \pgf@relevantforpicturesizefalse
7917     \pgfrememberpicturepositiononpagetrue
7918     \@@_qpoint:n { row - #1 }
7919     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7920     \@@_qpoint:n { col - #2 }
7921     \dim_set_eq:NN \l_tmpb_dim \pgf@x
7922     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7923     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7924     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7925     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7926     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7927     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7928 }

```

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.

```

7929     \CT@arc@
7930     \pgfsetroundcap
7931     \pgfusepathqstroke
7932 }
7933 \pgfset { inner~sep = 1 pt }
7934 \pgfscope
7935 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7936 \pgfnode { rectangle } { south~west }
7937 {
7938     \begin { minipage } { 20 cm }
7939     \@@_math_toggle_token: #5 \@@_math_toggle_token:
7940     \end { minipage }
7941 }
7942 { }
7943 { }
7944 \endpgfscope
7945 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7946 \pgfnode { rectangle } { north~east }
7947 {
7948     \begin { minipage } { 20 cm }
7949     \raggedleft
7950     \@@_math_toggle_token: #6 \@@_math_toggle_token:
7951     \end { minipage }
7952 }
7953 { }
7954 { }
7955 \endpgfpicture
7956 }

```

32 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 81.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
7957 \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 `\`.

```
7958 \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`.

```
7959 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7960 {
7961   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7962   \@@_CodeAfter_iv:n
7963 }
```

We catch the argument of the command `\end` (in `#1`).

```
7964 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7965 {
```

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.

```
7966   \str_if_eq:eeTF \@currenvir { #1 }
7967   { \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`.

```
7968   {
7969     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7970     \@@_CodeAfter_ii:n
7971   }
7972 }
```

33 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
7973 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7974 {
7975   \pgfpicture
7976   \pgfrememberpicturepositiononpagetrue
7977   \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.

```

7978 \@@_qpoint:n { row - 1 }
7979 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7980 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7981 \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).

```

7982 \bool_if:nTF { #3 }
7983 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7984 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7985 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7986 {
7987   \cs_if_exist:cT
7988   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7989   {
7990     \pgfpointanchor
7991     { \@@_env: - ##1 - #2 }
7992     { \bool_if:nTF { #3 } { west } { east } }
7993     \dim_set:Nn \l_tmpa_dim
7994     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7995   }
7996 }

```

Now we can put the delimiter with a node of PGF.

```

7997 \pgfset { inner~sep = \c_zero_dim }
7998 \dim_zero:N \nulldelimiterspace
7999 \pgftransformshift
8000 {
8001   \pgfpoint
8002   { \l_tmpa_dim }
8003   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8004 }
8005 \pgfnode
8006 { rectangle }
8007 { \bool_if:nTF { #3 } { east } { west } }
8008 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8009 \nullfont
8010 \c_math_toggle_token
8011 \@@_color:V \l_@@_delimiters_color_tl
8012 \bool_if:nTF { #3 } { \left #1 } { \left . }
8013 \vcenter
8014 {
8015   \nullfont
8016   \hrule \@height
8017   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8018   \@depth \c_zero_dim
8019   \@width \c_zero_dim
8020 }
8021 \bool_if:nTF { #3 } { \right . } { \right #1 }
8022 \c_math_toggle_token
8023 }
8024 { }
8025 { }
8026 \endpgfpicture
8027 }

```


34 The command \SubMatrix

```

8028 \keys_define:nn { NiceMatrix / sub-matrix }
8029 {
8030   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8031   extra-height .value_required:n = true ,
8032   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8033   left-xshift .value_required:n = true ,
8034   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8035   right-xshift .value_required:n = true ,
8036   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8037   xshift .value_required:n = true ,
8038   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8039   delimiters / color .value_required:n = true ,
8040   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8041   slim .default:n = true ,
8042   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8043   hlines .default:n = all ,
8044   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8045   vlines .default:n = all ,
8046   hvlines .meta:n = { hlines, vlines } ,
8047   hvlines .value_forbidden:n = true
8048 }
8049 \keys_define:nn { NiceMatrix }
8050 {
8051   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8052   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8053   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8054   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8055 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8056 \keys_define:nn { NiceMatrix / SubMatrix }
8057 {
8058   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8059   delimiters / color .value_required:n = true ,
8060   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8061   hlines .default:n = all ,
8062   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8063   vlines .default:n = all ,
8064   hvlines .meta:n = { hlines, vlines } ,
8065   hvlines .value_forbidden:n = true ,
8066   name .code:n =
8067     \tl_if_empty:nTF { #1 }
8068     { \@@_error:n { Invalid-name } }
8069     {
8070       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8071       {
8072         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8073         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8074         {
8075           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8076           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8077         }
8078       }
8079       { \@@_error:n { Invalid-name } }
8080     } ,
8081   name .value_required:n = true ,
8082   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8083   rules .value_required:n = true ,
8084   code .tl_set:N = \l_@@_code_tl ,

```

```

8085     code .value_required:n = true ,
8086     unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8087 }

8088 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8089 {
8090     \peek_remove_spaces:n
8091     {
8092         \tl_gput_right:Nx \g_@@_pre_code_after_tl
8093         {
8094             \SubMatrix { #1 } { #2 } { #3 } { #4 }
8095             [
8096                 delimiters / color = \l_@@_delimiters_color_tl ,
8097                 hlines = \l_@@_submatrix_hlines_clist ,
8098                 vlines = \l_@@_submatrix_vlines_clist ,
8099                 extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8100                 left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8101                 right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8102                 slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8103                 #5
8104             ]
8105         }
8106         \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8107     }
8108 }

8109 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8110 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8111 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8112 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8113 {
8114     \seq_gput_right:Nx \g_@@_submatrix_seq
8115     {
We use \str_if_eq:nnTF because it is fully expandable.
8116         { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8117         { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8118         { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8119         { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8120     }
8121 }

```

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.

```

8122 \hook_gput_code:nnn { begindocument } { . }
8123 {
8124     \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8125     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

8126 \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8127 {
8128   \peek_remove_spaces:n
8129   {
8130     \@@_sub_matrix:nnnnnnn
8131     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8132   }
8133 }
8134 }

```

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).

```

8135 \NewDocumentCommand \@@_compute_i_j:nn
8136 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8137 { \@@_compute_i_j:nnnn #1 #2 }
8138 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8139 {
8140   \tl_set:Nn \l_@@_first_i_tl { #1 }
8141   \tl_set:Nn \l_@@_first_j_tl { #2 }
8142   \tl_set:Nn \l_@@_last_i_tl { #3 }
8143   \tl_set:Nn \l_@@_last_j_tl { #4 }
8144   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8145   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8146   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8147   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8148   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8149   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8150   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8151   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8152 }
8153 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8154 {
8155   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8156 \@@_compute_i_j:nn { #2 } { #3 }
8157 % added 6.19b
8158 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8159 { \cs_set:Npn \arraystretch { 1 } }
8160 \bool_lazy_or:nnTF
8161 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8162 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8163 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8164 {
8165   \str_clear_new:N \l_@@_submatrix_name_str
8166   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8167   \pgfpicture
8168   \pgfrememberpicturerepositiononpagetrue
8169   \pgf@relevantforpicturesizefalse
8170   \pgfset { inner-sep = \c_zero_dim }
8171   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8172   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

8173 \bool_if:NTF \l_@@_submatrix_slim_bool
8174 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8175 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8176 {
8177   \cs_if_exist:cT
8178   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8179   {
8180     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }

```

```

8181         \dim_set:Nn \l_@@_x_initial_dim
8182         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8183     }
8184     \cs_if_exist:cT
8185     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8186     {
8187         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8188         \dim_set:Nn \l_@@_x_final_dim
8189         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8190     }
8191 }
8192 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8193 { \@@_error:nn { Impossible~delimiter } { left } }
8194 {
8195     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8196     { \@@_error:nn { Impossible~delimiter } { right } }
8197     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8198 }
8199 \endpgfpicture
8200 }
8201 \group_end:
8202 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8203 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8204 {
8205     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8206     \dim_set:Nn \l_@@_y_initial_dim
8207     {
8208         \fp_to_dim:n
8209         {
8210             \pgf@y
8211             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8212         }
8213     } % modified 6.13c
8214     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8215     \dim_set:Nn \l_@@_y_final_dim
8216     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8217     % modified 6.13c
8218     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8219     {
8220         \cs_if_exist:cT
8221         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8222         {
8223             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8224             \dim_set:Nn \l_@@_y_initial_dim
8225             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8226         }
8227         \cs_if_exist:cT
8228         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8229         {
8230             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8231             \dim_set:Nn \l_@@_y_final_dim
8232             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8233         }
8234     }
8235     \dim_set:Nn \l_tmpa_dim
8236     {
8237         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8238         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8239     }
8240     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8241 \group_begin:
8242 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8243 \@@_set_CT@arc@:V \l_@@_rules_color_tl
8244 \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`.

```

8245 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8246 {
8247   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8248   {
8249     \int_compare:nNnT
8250       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8251     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8252       \@@_qpoint:n { col - ##1 }
8253       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8254       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8255       \pgfusepathqstroke
8256     }
8257   }
8258 }

```

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.

```

8259 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
8260 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8261 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8262 {
8263   \bool_lazy_and:nnTF
8264     { \int_compare_p:nNn { ##1 } > 0 }
8265     {
8266       \int_compare_p:nNn
8267         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8268     {
8269       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8270       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8271       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8272       \pgfusepathqstroke
8273     }
8274     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8275 }

```

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.

```

8276 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
8277 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8278 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8279 {
8280   \bool_lazy_and:nnTF
8281     { \int_compare_p:nNn { ##1 } > 0 }
8282     {
8283       \int_compare_p:nNn
8284         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8285     {
8286       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8287 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8288      \dim_set:Nn \l_tmpa_dim
8289      { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8290      \str_case:nn { #1 }
8291      {
8292          ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8293            [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8294            \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8295          }
8296      \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8297      \dim_set:Nn \l_tmpb_dim
8298      { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8299      \str_case:nn { #2 }
8300      {
8301          ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8302          ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8303          \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8304      }
8305      \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8306      \pgfusepathqstroke
8307      \group_end:
8308  }
8309  { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8310  }

```

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).

```

8311      \str_if_empty:NF \l_@@_submatrix_name_str
8312      {
8313          \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8314          \l_@@_x_initial_dim \l_@@_y_initial_dim
8315          \l_@@_x_final_dim \l_@@_y_final_dim
8316      }
8317      \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`.

```

8318      \begin { pgfscope }
8319      \pgftransformshift
8320      {
8321          \pgfpoint
8322          { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8323          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8324      }
8325      \str_if_empty:NTF \l_@@_submatrix_name_str
8326      { \@@_node_left:nn #1 { } }
8327      { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8328      \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8329      \pgftransformshift
8330      {
8331          \pgfpoint
8332          { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8333          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8334      }
8335      \str_if_empty:NTF \l_@@_submatrix_name_str
8336      { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8337      {
8338          \@@_node_right:nnnn #2

```

```

8339         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8340     }

8341     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8342     \flag_clear_new:n { nicematrix }
8343     \l_@@_code_tl
8344 }

```

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`.

```

8345 \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.

```

8346 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8347 {
8348     \use:e
8349     { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8350 }

```

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 -.

```

8351 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8352 { #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.

```

8353 \tl_const:Nn \c_@@_integers_alist_tl
8354 {
8355     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8356     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8357     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8358     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8359 }

```

```

8360 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8361 {

```

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`.

```

8362     \tl_if_empty:nTF { #2 }
8363     {
8364         \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8365         {
8366             \flag_raise:n { nicematrix }
8367             \int_if_even:nTF { \flag_height:n { nicematrix } }
8368             { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8369             { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8370         }
8371         { #1 }
8372     }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8373     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8374   }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

8375 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8376 {
8377   \str_case:nnF { #1 }
8378   {
8379     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8380     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8381   }

```

Now the case of a node of the form $i-j$.

```

8382   {
8383     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8384     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8385   }
8386 }

```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

8387 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8388 {
8389   \pgfnode
8390   { rectangle }
8391   { east }
8392   {
8393     \nullfont
8394     \c_math_toggle_token
8395     \@@_color:V \l_@@_delimiters_color_tl
8396     \left #1
8397     \vcenter
8398     {
8399       \nullfont
8400       \hrule \@height \l_tmpa_dim
8401               \@depth \c_zero_dim
8402               \@width \c_zero_dim
8403     }
8404     \right .
8405     \c_math_toggle_token
8406   }
8407   { #2 }
8408   { }
8409 }

```

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.

```

8410 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8411 {
8412   \pgfnode
8413   { rectangle }
8414   { west }
8415   {
8416     \nullfont
8417     \c_math_toggle_token
8418     \@@_color:V \l_@@_delimiters_color_tl
8419     \left .
8420     \vcenter

```



```

8421     {
8422         \nullfont
8423         \hrule \@height \l_tmpa_dim
8424             \@depth \c_zero_dim
8425             \@width \c_zero_dim
8426     }
8427     \right #1
8428     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8429     ~ { \smash { #4 } }
8430     \c_math_toggle_token
8431 }
8432 { #2 }
8433 { }
8434 }

```

35 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

8435 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8436 {
8437     \peek_remove_spaces:n
8438     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8439 }
8440 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8441 {
8442     \peek_remove_spaces:n
8443     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8444 }
8445 \keys_define:nn { NiceMatrix / Brace }
8446 {
8447     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8448     left-shorten .default:n = true ,
8449     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8450     shorten .meta:n = { left-shorten , right-shorten } ,
8451     right-shorten .default:n = true ,
8452     yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8453     yshift .value_required:n = true ,
8454     yshift .initial:n = \c_zero_dim ,
8455     color .tl_set:N = \l_tmpa_tl ,
8456     color .value_required:n = true ,
8457     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8458 }

```

#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*.

```

8459 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8460 {
8461     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8462 \@@_compute_i_j:nn { #1 } { #2 }
8463 \bool_lazy_or:nnTF
8464 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8465 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8466 {
8467     \str_if_eq:nnTF { #5 } { under }

```

```

8468     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8469     { \@@_error:nn { Construct~too~large } { \OverBrace } }
8470 }
8471 {
8472   \tl_clear:N \l_tmpa_tl
8473   \keys_set:nn { NiceMatrix / Brace } { #4 }
8474   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8475   \pgfpicture
8476   \pgfrememberpicturepositiononpagetrue
8477   \pgf@relevantforpicturesizefalse
8478   \bool_if:NT \l_@@_brace_left_shorten_bool
8479   {
8480     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8481     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8482     {
8483       \cs_if_exist:cT
8484       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8485       {
8486         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8487         \dim_set:Nn \l_@@_x_initial_dim
8488         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8489       }
8490     }
8491   }
8492   \bool_lazy_or:nnT
8493   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8494   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8495   {
8496     \@@_qpoint:n { col - \l_@@_first_j_tl }
8497     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8498   }
8499   \bool_if:NT \l_@@_brace_right_shorten_bool
8500   {
8501     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8502     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8503     {
8504       \cs_if_exist:cT
8505       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8506       {
8507         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8508         \dim_set:Nn \l_@@_x_final_dim
8509         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8510       }
8511     }
8512   }
8513   \bool_lazy_or:nnT
8514   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8515   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8516   {
8517     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8518     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8519   }
8520   \pgfset { inner-sep = \c_zero_dim }
8521   \str_if_eq:nnTF { #5 } { under }
8522   { \@@_underbrace_i:n { #3 } }
8523   { \@@_overbrace_i:n { #3 } }
8524   \endpgfpicture
8525 }
8526 \group_end:
8527 }

```

The argument is the text to put above the brace.

```

8528 \cs_new_protected:Npn \@@_overbrace_i:n #1
8529 {

```

```

8530 \@@_qpoint:n { row - \l_@@_first_i_tl }
8531 \pgftransformshift
8532 {
8533   \pgfpoint
8534   { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8535   { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8536 }
8537 \pgfnode
8538 { rectangle }
8539 { south }
8540 {
8541   \vbox_top:n
8542   {
8543     \group_begin:
8544     \everycr { }
8545     \halign
8546     {
8547       \hfil ## \hfil \crr
8548       \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8549       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8550       \c_math_toggle_token
8551       \overbrace
8552       {
8553         \hbox_to_wd:nn
8554         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8555         { }
8556       }
8557       \c_math_toggle_token
8558       \cr
8559     }
8560     \group_end:
8561   }
8562 }
8563 { }
8564 { }
8565 }

```

The argument is the text to put under the brace.

```

8566 \cs_new_protected:Npn \@@_underbrace_i:n #1
8567 {
8568   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8569   \pgftransformshift
8570   {
8571     \pgfpoint
8572     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8573     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8574   }
8575   \pgfnode
8576   { rectangle }
8577   { north }
8578   {
8579     \group_begin:
8580     \everycr { }
8581     \vbox:n
8582     {
8583       \halign
8584       {
8585         \hfil ## \hfil \crr
8586         \c_math_toggle_token
8587         \underbrace
8588         {
8589           \hbox_to_wd:nn
8590           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8591           { }

```

```

8592         }
8593         \c_math_toggle_token
8594         \cr
8595         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8596         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8597     }
8598 }
8599 \group_end:
8600 }
8601 { }
8602 { }
8603 }

```

36 The command TikzEveryCell

```

8604 \bool_new:N \l_@@_not_empty_bool
8605 \bool_new:N \l_@@_empty_bool
8606
8607 \keys_define:nn { NiceMatrix / TikzEveryCell }
8608 {
8609     not-empty .code:n =
8610         \bool_lazy_or:nnTF
8611             \l_@@_in_code_after_bool
8612             \g_@@_recreate_cell_nodes_bool
8613             { \bool_set_true:N \l_@@_not_empty_bool }
8614             { \@@_error:n { detection-of~empty-cells } } ,
8615     not-empty .value_forbidden:n = true ,
8616     empty .code:n =
8617         \bool_lazy_or:nnTF
8618             \l_@@_in_code_after_bool
8619             \g_@@_recreate_cell_nodes_bool
8620             { \bool_set_true:N \l_@@_empty_bool }
8621             { \@@_error:n { detection-of~empty~cells } } ,
8622     empty .value_forbidden:n = true ,
8623     unknown .code:n = \@@_error:n { Unknown-key-for~TikzEveryCell }
8624 }
8625
8626
8627 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8628 {
8629     \IfPackageLoadedTF { tikz }
8630     {
8631         \group_begin:
8632         \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.

```

8633         \tl_set:Nn \l_tmpa_tl { { #2 } }
8634         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8635             { \@@_for_a_block:nnnnn ##1 }
8636         \@@_all_the_cells:
8637         \group_end:
8638     }
8639     { \@@_error:n { TikzEveryCell~without~tikz } }
8640 }
8641
8642 \tl_new:N \@@_i_tl
8643 \tl_new:N \@@_j_tl
8644
8645 \cs_new_protected:Nn \@@_all_the_cells:

```

```

8646 {
8647   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8648   {
8649     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8650     {
8651       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8652       {
8653         \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8654         { \@@_i_tl - \@@_j_tl }
8655         {
8656           \bool_set_false:N \l_tmpa_bool
8657           \cs_if_exist:cTF
8658           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8659           {
8660             \bool_if:NF \l_@@_empty_bool
8661             { \bool_set_true:N \l_tmpa_bool }
8662           }
8663           {
8664             \bool_if:NF \l_@@_not_empty_bool
8665             { \bool_set_true:N \l_tmpa_bool }
8666           }
8667           \bool_if:NT \l_tmpa_bool
8668           {
8669             \@@_block_tikz:nnnnV
8670             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8671           }
8672         }
8673       }
8674     }
8675   }
8676 }
8677
8678 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8679 {
8680   \bool_if:NF \l_@@_empty_bool
8681   {
8682     \@@_block_tikz:nnnnV
8683     { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8684   }
8685   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8686 }
8687
8688 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8689 {
8690   \int_step_inline:nnn { #1 } { #3 }
8691   {
8692     \int_step_inline:nnn { #2 } { #4 }
8693     { \cs_set:cpn { cell - ##1 - ####1 } { } }
8694   }
8695 }

```

37 The command \ShowCellNames

```

8696 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8697 {
8698   \dim_zero_new:N \g_@@_tmpc_dim
8699   \dim_zero_new:N \g_@@_tmpd_dim
8700   \dim_zero_new:N \g_@@_tmpe_dim
8701   \int_step_inline:nn \c@iRow
8702   {
8703     \begin { pgfpicture }
8704     \@@_qpoint:n { row - ##1 }

```

```

8705 \dim_set_eq:Nn \l_tmpa_dim \pgf@y
8706 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8707 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8708 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8709 \bool_if:NTF \l_@@_in_code_after_bool
8710 \end { pgfpicture }
8711 \int_step_inline:nn \c@jCol
8712 {
8713   \hbox_set:Nn \l_tmpa_box
8714   { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
8715   \begin { pgfpicture }
8716     \@@_qpoint:n { col - ####1 }
8717     \dim_gset_eq:Nn \g_@@_tmpc_dim \pgf@x
8718     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8719     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8720     \dim_gset_eq:Nn \g_@@_tmpe_dim \pgf@x
8721     \endpgfpicture
8722   \end { pgfpicture }
8723   \fp_set:Nn \l_tmpa_fp
8724   {
8725     \fp_min:nn
8726     {
8727       \fp_min:nn
8728       { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8729       { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8730     }
8731     { 1.0 }
8732   }
8733   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8734   \pgfpicture
8735   \pgfrememberpicturepositiononpagetrue
8736   \pgf@relevantforpicturesizefalse
8737   \pgftransformshift
8738   {
8739     \pgfpoint
8740     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8741     { \dim_use:N \g_tmpa_dim }
8742   }
8743   \pgfnode
8744   { rectangle }
8745   { center }
8746   { \box_use:N \l_tmpa_box }
8747   { }
8748   { }
8749   \endpgfpicture
8750 }
8751 }
8752 }
8753 \NewDocumentCommand \@@_ShowCellNames { }
8754 {
8755   \bool_if:NT \l_@@_in_code_after_bool
8756   {
8757     \pgfpicture
8758     \pgfrememberpicturepositiononpagetrue
8759     \pgf@relevantforpicturesizefalse
8760     \pgfpathrectanglecorners
8761     { \@@_qpoint:n { 1 } }
8762     {
8763       \@@_qpoint:n
8764       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8765     }
8766     \pgfsetfillopacity { 0.75 }
8767     \pgfsetfillcolor { white }

```

```

8768     \pgfusepathqfill
8769     \endpgfpicture
8770 }
8771 \dim_zero_new:N \g_@@_tmpc_dim
8772 \dim_zero_new:N \g_@@_tmpd_dim
8773 \dim_zero_new:N \g_@@_tmpe_dim
8774 \int_step_inline:nn \c@iRow
8775 {
8776     \bool_if:NTF \l_@@_in_code_after_bool
8777     {
8778         \pgfpicture
8779         \pgfrememberpicturepositiononpagetrue
8780         \pgf@relevantforpicturesizefalse
8781     }
8782     { \begin { pgfpicture } }
8783     \@@_qpoint:n { row - ##1 }
8784     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8785     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8786     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8787     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8788     \bool_if:NTF \l_@@_in_code_after_bool
8789     { \endpgfpicture }
8790     { \end { pgfpicture } }
8791     \int_step_inline:nn \c@jCol
8792     {
8793         \hbox_set:Nn \l_tmpa_box
8794         {
8795             \normalfont \Large \sffamily \bfseries
8796             \bool_if:NTF \l_@@_in_code_after_bool
8797             { \color { red } }
8798             { \color { red ! 50 } }
8799             ##1 - #####1
8800         }
8801         \bool_if:NTF \l_@@_in_code_after_bool
8802         {
8803             \pgfpicture
8804             \pgfrememberpicturepositiononpagetrue
8805             \pgf@relevantforpicturesizefalse
8806         }
8807         { \begin { pgfpicture } }
8808         \@@_qpoint:n { col - #####1 }
8809         \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8810         \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8811         \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8812         \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8813         \bool_if:NTF \l_@@_in_code_after_bool
8814         { \endpgfpicture }
8815         { \end { pgfpicture } }
8816         \fp_set:Nn \l_tmpa_fp
8817         {
8818             \fp_min:nn
8819             {
8820                 \fp_min:nn
8821                 { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8822                 { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8823             }
8824             { 1.0 }
8825         }
8826         \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8827         \pgfpicture
8828         \pgfrememberpicturepositiononpagetrue
8829         \pgf@relevantforpicturesizefalse
8830         \pgftransformshift

```

```

8831         {
8832             \pgfpoint
8833             { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8834             { \dim_use:N \g_tmpa_dim }
8835         }
8836     \pgfnode
8837     { rectangle }
8838     { center }
8839     { \box_use:N \l_tmpa_box }
8840     { }
8841     { }
8842 \endpgfpicture
8843 }
8844 }
8845 }

```

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.

```

8846 \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.

```

8847 \bool_new:N \g_@@_footnote_bool
8848 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
8849 {
8850     The~key~'\l_keys_key_str'~is~unknown. \\
8851     That~key~will~be~ignored. \\
8852     For~a~list~of~the~available~keys,~type~H~<return>.
8853 }
8854 {
8855     The~available~keys~are~(in~alphabetic~order):~
8856     footnote,~
8857     footnotehyper,~
8858     messages-for-Overleaf,~
8859     no-test-for-array,~
8860     renew-dots,~and~
8861     renew-matrix.
8862 }
8863 \keys_define:nn { NiceMatrix / Package }
8864 {
8865     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8866     renew-dots .value_forbidden:n = true ,
8867     renew-matrix .code:n = \@@_renew_matrix: ,
8868     renew-matrix .value_forbidden:n = true ,
8869     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
8870     footnote .bool_set:N = \g_@@_footnote_bool ,
8871     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
8872     no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
8873     no-test-for-array .default:n = true ,
8874     unknown .code:n = \@@_error:n { Unknown~key~for~package }
8875 }
8876 \ProcessKeysOptions { NiceMatrix / Package }

```



```

8877 \@@_msg_new:nn { footnote-with-footnotehyper-package }
8878 {
8879   You~can't~use~the~option~'footnote'~because~the~package~
8880   footnotehyper~has~already~been~loaded.~
8881   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8882   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8883   of~the~package~footnotehyper.\\
8884   The~package~footnote~won't~be~loaded.
8885 }
8886 \@@_msg_new:nn { footnotehyper-with-footnote-package }
8887 {
8888   You~can't~use~the~option~'footnotehyper'~because~the~package~
8889   footnote~has~already~been~loaded.~
8890   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8891   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8892   of~the~package~footnote.\\
8893   The~package~footnotehyper~won't~be~loaded.
8894 }

```

```

8895 \bool_if:NT \g_@@_footnote_bool
8896 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8897   \IfClassLoadedTF { beamer }
8898   { \bool_set_false:N \g_@@_footnote_bool }
8899   {
8900     \IfPackageLoadedTF { footnotehyper }
8901     { \@@_error:n { footnote-with-footnotehyper-package } }
8902     { \usepackage { footnote } }
8903   }
8904 }
8905 \bool_if:NT \g_@@_footnotehyper_bool
8906 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8907   \IfClassLoadedTF { beamer }
8908   { \bool_set_false:N \g_@@_footnote_bool }
8909   {
8910     \IfPackageLoadedTF { footnote }
8911     { \@@_error:n { footnotehyper-with-footnote-package } }
8912     { \usepackage { footnotehyper } }
8913   }
8914   \bool_set_true:N \g_@@_footnote_bool
8915 }

```

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.

```

8916 \bool_new:N \l_@@_underscore_loaded_bool
8917 \IfPackageLoadedTF { underscore }
8918 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8919 { }

```

```

8920 \hook_gput_code:nnn { begindocument } { . }
8921 {
8922   \bool_if:NF \l_@@_underscore_loaded_bool
8923   {
8924     \IfPackageLoadedTF { underscore }
8925     { \@@_error:n { underscore~after~nicematrix } }
8926     { }
8927   }
8928 }

```

40 Error messages of the package

```

8929 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
8930 { \str_const:Nn \c_@@_available_keys_str { } }
8931 {
8932   \str_const:Nn \c_@@_available_keys_str
8933   { For~a~list~of~the~available~keys,~type~H~<return>. }
8934 }

8935 \seq_new:N \g_@@_types_of_matrix_seq
8936 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8937 {
8938   NiceMatrix ,
8939   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8940 }
8941 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8942 { \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`.

```

8943 \cs_new_protected:Npn \@@_error_too_much_cols:
8944 {
8945   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8946   {
8947     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8948     { \@@_fatal:n { too~much~cols~for~matrix } }
8949     {
8950       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8951       { \@@_fatal:n { too~much~cols~for~matrix } }
8952       {
8953         \bool_if:NF \l_@@_last_col_without_value_bool
8954         { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
8955       }
8956     }
8957   }
8958   { \@@_fatal:nn { too~much~cols~for~array } }
8959 }

```

The following command must *not* be protected since it's used in an error message.

```

8960 \cs_new:Npn \@@_message_hdotsfor:
8961 {
8962   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8963   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
8964 }

8965 \@@_msg_new:nn { hvlines,~rounded~corners~and~corners }
8966 {
8967   Incompatible~options.\

```

```

8968     You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
8969     The~output~will~not~be~reliable.
8970 }

8971 \@@_msg_new:nn { negative~weight }
8972 {
8973     Negative~weight.\\
8974     The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
8975     the~value~'\int_use:N \l_@@_weight_int'.\\
8976     The~absolute~value~will~be~used.
8977 }

8978 \@@_msg_new:nn { last~col~not~used }
8979 {
8980     Column~not~used.\\
8981     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
8982     in~your~\@@_full_name_env:~However,~you~can~go~on.
8983 }

8984 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
8985 {
8986     Too~much~columns.\\
8987     In~the~row~\int_eval:n { \c@iRow },~
8988     you~try~to~use~more~columns~
8989     than~allowed~by~your~\@@_full_name_env:~\@@_message_hdotsfor:\\
8990     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
8991     (plus~the~exterior~columns).~This~error~is~fatal.
8992 }

8993 \@@_msg_new:nn { too~much~cols~for~matrix }
8994 {
8995     Too~much~columns.\\
8996     In~the~row~\int_eval:n { \c@iRow },~
8997     you~try~to~use~more~columns~than~allowed~by~your~
8998     \@@_full_name_env:~\@@_message_hdotsfor:\\ Recall~that~the~maximal~
8999     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9000     columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9001     Its~current~value~is~\int_use:N \c@MaxMatrixCols\\ (use~
9002     \token_to_str:N \setcounter\\ to~change~that~value).~
9003     This~error~is~fatal.
9004 }

9005 \@@_msg_new:nn { too~much~cols~for~array }
9006 {
9007     Too~much~columns.\\
9008     In~the~row~\int_eval:n { \c@iRow },~
9009     ~you~try~to~use~more~columns~than~allowed~by~your~
9010     \@@_full_name_env:~\@@_message_hdotsfor:\\ The~maximal~number~of~columns~is~
9011     \int_use:N \g_@@_static_num_of_col_int\\
9012     ~(plus~the~potential~exterior~ones).
9013     This~error~is~fatal.
9014 }

9015 \@@_msg_new:nn { columns~not~used }
9016 {
9017     Columns~not~used.\\
9018     The~preamble~of~your~\@@_full_name_env:\\ announces~\int_use:N
9019     \g_@@_static_num_of_col_int\\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9020     The~columns~you~did~not~used~won't~be~created.\\
9021     You~won't~have~similar~error~till~the~end~of~the~document.
9022 }

9023 \@@_msg_new:nn { in~first~col }
9024 {
9025     Erroneous~use.\\
9026     You~can't~use~the~command~\#1 in~the~first~column~(number~0)~of~the~array.\\
9027     That~command~will~be~ignored.

```

```

9028 }
9029 \@@_msg_new:nn { in~last~col }
9030 {
9031   Erroneous~use.\\
9032   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9033   That~command~will~be~ignored.
9034 }
9035 \@@_msg_new:nn { in~first~row }
9036 {
9037   Erroneous~use.\\
9038   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9039   That~command~will~be~ignored.
9040 }
9041 \@@_msg_new:nn { in~last~row }
9042 {
9043   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9044   That~command~will~be~ignored.
9045 }
9046 \@@_msg_new:nn { caption~outside~float }
9047 {
9048   Key~caption~forbidden.\\
9049   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9050   environment.~This~key~will~be~ignored.
9051 }
9052 \@@_msg_new:nn { short~caption~without~caption }
9053 {
9054   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9055   However,~your~'short~caption'~will~be~used~as~'caption'.
9056 }
9057 \@@_msg_new:nn { double~closing~delimiter }
9058 {
9059   Double~delimiter.\\
9060   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9061   delimiter.~This~delimiter~will~be~ignored.
9062 }
9063 \@@_msg_new:nn { delimiter~after~opening }
9064 {
9065   Double~delimiter.\\
9066   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9067   delimiter.~That~delimiter~will~be~ignored.
9068 }
9069 \@@_msg_new:nn { bad~option~for~line~style }
9070 {
9071   Bad~line~style.\\
9072   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9073   is~'standard'.~That~key~will~be~ignored.
9074 }
9075 \@@_msg_new:nn { Identical~notes~in~caption }
9076 {
9077   Identical~tabular~notes.\\
9078   You~can't~put~several~notes~with~the~same~content~in~
9079   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9080   If~you~go~on,~the~output~will~probably~be~erroneous.
9081 }
9082 \@@_msg_new:nn { tabularnote~below~the~tabular }
9083 {
9084   \token_to_str:N \tabularnote\ forbidden\\
9085   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9086   of~your~tabular~because~the~caption~will~be~composed~below~

```

```

9087 the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9088 key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9089 Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9090 no~similar~error~will~raised~in~this~document.
9091 }
9092 \@@_msg_new:nn { Unknown~key~for~rules }
9093 {
9094   Unknown~key.\\
9095   There~is~only~two~keys~available~here:~width~and~color.\\
9096   Your~key~'\l_keys_key_str'~will~be~ignored.
9097 }
9098 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9099 {
9100   Unknown~key.\\
9101   There~is~only~two~keys~available~here:~
9102   'empty'~and~'not~empty'.\\
9103   Your~key~'\l_keys_key_str'~will~be~ignored.
9104 }
9105 \@@_msg_new:nn { Unknown~key~for~rotate }
9106 {
9107   Unknown~key.\\
9108   The~only~key~available~here~is~'c'.\\
9109   Your~key~'\l_keys_key_str'~will~be~ignored.
9110 }
9111 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9112 {
9113   Unknown~key.\\
9114   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9115   It~you~go~on,~you~will~probably~have~other~errors. \\
9116   \c_@@_available_keys_str
9117 }
9118 {
9119   The~available~keys~are~(in~alphabetic~order):~
9120   ccommand,~
9121   color,~
9122   command,~
9123   dotted,~
9124   letter,~
9125   multiplicity,~
9126   sep~color,~
9127   tikz,~and~total~width.
9128 }
9129 \@@_msg_new:nnn { Unknown~key~for~xdots }
9130 {
9131   Unknown~key.\\
9132   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9133   \c_@@_available_keys_str
9134 }
9135 {
9136   The~available~keys~are~(in~alphabetic~order):~
9137   'color',~
9138   'horizontal~labels',~
9139   'inter',~
9140   'line~style',~
9141   'radius',~
9142   'shorten',~
9143   'shorten~end'~and~'shorten~start'.
9144 }
9145 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9146 {
9147   Unknown~key.\\

```

```

9148     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9149     (and~you~try~to~use~'\l_keys_key_str')\\
9150     That~key~will~be~ignored.
9151 }

9152 \@@_msg_new:nn { label~without~caption }
9153 {
9154     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9155     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9156 }

9157 \@@_msg_new:nn { W~warning }
9158 {
9159     Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
9160     (row~\int_use:N \c@iRow).
9161 }

9162 \@@_msg_new:nn { Construct~too~large }
9163 {
9164     Construct~too~large.\\
9165     Your~command~\token_to_str:N #1
9166     can't~be~drawn~because~your~matrix~is~too~small.\\
9167     That~command~will~be~ignored.
9168 }

9169 \@@_msg_new:nn { underscore~after~nicematrix }
9170 {
9171     Problem~with~'underscore'.\\
9172     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9173     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9174     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9175 }

9176 \@@_msg_new:nn { ampersand~in~light~syntax }
9177 {
9178     Ampersand~forbidden.\\
9179     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9180     the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9181 }

9182 \@@_msg_new:nn { double~backslash~in~light~syntax }
9183 {
9184     Double~backslash~forbidden.\\
9185     You~can't~use~\token_to_str:N
9186     \\~to~separate~rows~because~the~key~'light-syntax'~
9187     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9188     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9189 }

9190 \@@_msg_new:nn { hlines~with~color }
9191 {
9192     Incompatible~keys.\\
9193     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9194     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9195     Maybe~it~will~possible~in~future~version.\\
9196     Your~key~will~be~discarded.
9197 }

9198 \@@_msg_new:nn { bad~value~for~baseline }
9199 {
9200     Bad~value~for~baseline.\\
9201     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9202     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9203     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9204     the~form~'line-i'.\\
9205     A~value~of~1~will~be~used.
9206 }

9207 \@@_msg_new:nn { detection~of~empty~cells }

```

```

9208 {
9209     Problem~with~'not-empty'\
9210     For~technical~reasons,~you~must~activate~
9211     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9212     in~order~to~use~the~key~'\l_keys_key_str'.\
9213     That~key~will~be~ignored.
9214 }
9215 \@@_msg_new:nn { ragged2e~not~loaded }
9216 {
9217     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9218     your~column~'\l_@@_vpos_col_str'~(or~'X')~.~The~key~'\str_lowercase:V
9219     \l_keys_key_str'~will~be~used~instead.
9220 }
9221 \@@_msg_new:nn { Invalid~name }
9222 {
9223     Invalid~name.\
9224     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9225     \SubMatrix\ of~your~\@@_full_name_env:.\
9226     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
9227     This~key~will~be~ignored.
9228 }
9229 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9230 {
9231     Wrong~line.\
9232     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9233     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9234     number~is~not~valid.~It~will~be~ignored.
9235 }
9236 \@@_msg_new:nn { Impossible~delimiter }
9237 {
9238     Impossible~delimiter.\
9239     It's~impossible~to~draw~the~#1~delimiter~of~your~
9240     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9241     in~that~column.
9242     \bool_if:NT \l_@@_submatrix_slim_bool
9243     { ~Maybe~you~should~try~without~the~key~'slim'. } \
9244     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9245 }
9246 \@@_msg_new:nnn { width~without~X~columns }
9247 {
9248     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9249     That~key~will~be~ignored.
9250 }
9251 {
9252     This~message~is~the~message~'width~without~X~columns'~
9253     of~the~module~'nicematrix'.~
9254     The~experimented~users~can~disable~that~message~with~
9255     \token_to_str:N \msg_redirect_name:nnn.\
9256 }
9257
9258 \@@_msg_new:nn { key~multiplicity~with~dotted }
9259 {
9260     Incompatible~keys. \
9261     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9262     in~a~'custom-line'.~They~are~incompatible. \
9263     The~key~'multiplicity'~will~be~discarded.
9264 }
9265 \@@_msg_new:nn { empty~environment }
9266 {
9267     Empty~environment.\
9268     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.

```

```

9269 }
9270 \@@_msg_new:nn { No-letter~and~no-command }
9271 {
9272   Erroneous-use.\
9273   Your-use-of~'custom-line'~is-no-op~since~you~don't~have~used~the~
9274   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9275   ~'ccommand'~(to~draw~horizontal~rules).\
9276   However,~you~can~go~on.
9277 }
9278 \@@_msg_new:nn { Forbidden-letter }
9279 {
9280   Forbidden-letter.\
9281   You~can't~use~the~letter~'#1'~for~a~customized~line.\
9282   It~will~be~ignored.
9283 }
9284 \@@_msg_new:nn { Several-letters }
9285 {
9286   Wrong-name.\
9287   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9288   have~used~'\l_@@_letter_str').\
9289   It~will~be~ignored.
9290 }
9291 \@@_msg_new:nn { Delimiter~with~small }
9292 {
9293   Delimiter~forbidden.\
9294   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9295   because~the~key~'small'~is~in~force.\
9296   This~error~is~fatal.
9297 }
9298 \@@_msg_new:nn { unknown-cell~for~line~in~CodeAfter }
9299 {
9300   Unknown~cell.\
9301   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9302   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9303   can't~be~executed~because~a~cell~doesn't~exist.\
9304   This~command~\token_to_str:N \line\ will~be~ignored.
9305 }
9306 \@@_msg_new:nnn { Duplicate-name~for~SubMatrix }
9307 {
9308   Duplicate-name.\
9309   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9310   in~this~\@@_full_name_env:.\
9311   This~key~will~be~ignored.\
9312   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9313     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9314 }
9315 {
9316   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9317   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9318 }
9319 \@@_msg_new:nn { r~or~l~with~preamble }
9320 {
9321   Erroneous-use.\
9322   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9323   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9324   your~\@@_full_name_env:.\
9325   This~key~will~be~ignored.
9326 }
9327 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9328 {

```



```

9329 Erroneous-use.\
9330 You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9331 the~array.~This~error~is~fatal.
9332 }

9333 \@@_msg_new:nn { bad-corner }
9334 {
9335   Bad-corner.\
9336   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9337   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
9338   This~specification~of~corner~will~be~ignored.
9339 }

9340 \@@_msg_new:nn { bad-border }
9341 {
9342   Bad-border.\
9343   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9344   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9345   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9346   also~use~the~key~'tikz'
9347   \IfPackageLoadedTF { tikz }
9348     { }
9349     {~if~you~load~the~LaTeX~package~'tikz'}).\
9350   This~specification~of~border~will~be~ignored.
9351 }

9352 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9353 {
9354   TikZ~not~loaded.\
9355   You~can't~use~\token_to_str:N \TikzEveryCell\
9356   because~you~have~not~loaded~tikz.~
9357   This~command~will~be~ignored.
9358 }

9359 \@@_msg_new:nn { tikz~key~without~tikz }
9360 {
9361   TikZ~not~loaded.\
9362   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9363   \Block'~because~you~have~not~loaded~tikz.~
9364   This~key~will~be~ignored.
9365 }

9366 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
9367 {
9368   Erroneous-use.\
9369   In~the~\@@_full_name_env:,~you~must~use~the~key~
9370   'last-col'~without~value.\
9371   However,~you~can~go~on~for~this~time~
9372   (the~value~'\l_keys_value_tl'~will~be~ignored).
9373 }

9374 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
9375 {
9376   Erroneous-use.\
9377   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9378   'last-col'~without~value.\
9379   However,~you~can~go~on~for~this~time~
9380   (the~value~'\l_keys_value_tl'~will~be~ignored).
9381 }

9382 \@@_msg_new:nn { Block~too~large~1 }
9383 {
9384   Block~too~large.\
9385   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
9386   too~small~for~that~block. \
9387 }

9388 \@@_msg_new:nn { Block~too~large~2 }

```

```

9389 {
9390   Block-too-large.\
9391   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9392   \g_@@_static_num_of_col_int\
9393   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9394   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
9395   (&)~at~the~end~of~the~first~row~of~your~
9396   \@@_full_name_env:.\
9397   This~block~and~maybe~others~will~be~ignored.
9398 }
9399 \@@_msg_new:nn { unknown~column~type }
9400 {
9401   Bad~column~type.\
9402   The~column~type~'#1'~in~your~\@@_full_name_env:\
9403   is~unknown. \
9404   This~error~is~fatal.
9405 }
9406 \@@_msg_new:nn { unknown~column~type~S }
9407 {
9408   Bad~column~type.\
9409   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \
9410   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9411   load~that~package. \
9412   This~error~is~fatal.
9413 }
9414 \@@_msg_new:nn { tabularnote~forbidden }
9415 {
9416   Forbidden~command.\
9417   You~can't~use~the~command~\token_to_str:N\tabularnote\
9418   ~here.~This~command~is~available~only~in~
9419   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9420   the~argument~of~a~command~\token_to_str:N \caption\ included~
9421   in~an~environment~{table}. \
9422   This~command~will~be~ignored.
9423 }
9424 \@@_msg_new:nn { borders~forbidden }
9425 {
9426   Forbidden~key.\
9427   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9428   because~the~option~'rounded~corners'~
9429   is~in~force~with~a~non~zero~value.\
9430   This~key~will~be~ignored.
9431 }
9432 \@@_msg_new:nn { bottomrule~without~booktabs }
9433 {
9434   booktabs~not~loaded.\
9435   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9436   loaded~'booktabs'.\
9437   This~key~will~be~ignored.
9438 }
9439 \@@_msg_new:nn { enumitem~not~loaded }
9440 {
9441   enumitem~not~loaded.\
9442   You~can't~use~the~command~\token_to_str:N\tabularnote\
9443   ~because~you~haven't~loaded~'enumitem'.\
9444   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9445   ignored~in~the~document.
9446 }
9447 \@@_msg_new:nn { tikz~without~tikz }
9448 {
9449   Tikz~not~loaded.\

```

```

9450     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9451     loaded.~If~you~go~on,~that~key~will~be~ignored.
9452 }

9453 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9454 {
9455     Tikz~not~loaded.\\
9456     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9457     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9458     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9459     use~that~custom~line.
9460 }

9461 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9462 {
9463     Tikz~not~loaded.\\
9464     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9465     command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9466     That~key~will~be~ignored.
9467 }

9468 \@@_msg_new:nn { without~color~inside }
9469 {
9470     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9471     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9472     outside~\token_to_str:N \CodeBefore,~you~
9473     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:~\\
9474     You~can~go~on~but~you~may~need~more~compilations.
9475 }

9476 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9477 {
9478     Erroneous~use.\\
9479     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9480     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9481     The~key~'color'~will~be~discarded.
9482 }

9483 \@@_msg_new:nn { Wrong~last~row }
9484 {
9485     Wrong~number.\\
9486     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9487     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9488     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9489     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9490     without~value~(more~compilations~might~be~necessary).
9491 }

9492 \@@_msg_new:nn { Yet~in~env }
9493 {
9494     Nested~environments.\\
9495     Environments~of~nicematrix~can't~be~nested.\\
9496     This~error~is~fatal.
9497 }

9498 \@@_msg_new:nn { Outside~math~mode }
9499 {
9500     Outside~math~mode.\\
9501     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9502     (and~not~in~\token_to_str:N \vcenter).\\
9503     This~error~is~fatal.
9504 }

9505 \@@_msg_new:nn { One~letter~allowed }
9506 {
9507     Bad~name.\\
9508     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9509     It~will~be~ignored.

```

```

9510 }
9511 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9512 {
9513   Environment~{TabularNote}~forbidden.\
9514   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9515   but~*before*~the~\token_to_str:N \CodeAfter.\
9516   This~environment~{TabularNote}~will~be~ignored.
9517 }
9518 \@@_msg_new:nn { varwidth~not~loaded }
9519 {
9520   varwidth~not~loaded.\
9521   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9522   loaded.\
9523   Your~column~will~behave~like~'p'.
9524 }
9525 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
9526 {
9527   Unknown~key.\
9528   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
9529   \c_@@_available_keys_str
9530 }
9531 {
9532   The~available~keys~are~(in~alphabetic~order):~
9533   color,~
9534   dotted,~
9535   multiplicity,~
9536   sep-color,~
9537   tikz,~and~total-width.
9538 }
9539
9540 \@@_msg_new:nnn { Unknown~key~for~Block }
9541 {
9542   Unknown~key.\
9543   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9544   \Block.\ It~will~be~ignored. \
9545   \c_@@_available_keys_str
9546 }
9547 {
9548   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9549   hlines,~hvlines,~l,~line-width,~name,~opacity,~rounded-corners,~r,~
9550   respect-arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9551 }
9552 \@@_msg_new:nnn { Unknown~key~for~Brace }
9553 {
9554   Unknown~key.\
9555   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9556   \UnderBrace\ and~\token_to_str:N \OverBrace.\
9557   It~will~be~ignored. \
9558   \c_@@_available_keys_str
9559 }
9560 {
9561   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
9562   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
9563   right-shorten)~and~yshift.
9564 }
9565 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9566 {
9567   Unknown~key.\
9568   The~key~'\l_keys_key_str'~is~unknown.\
9569   It~will~be~ignored. \
9570   \c_@@_available_keys_str

```

```

9571 }
9572 {
9573   The~available~keys~are~(in~alphabetic~order):~
9574   delimiters/color,~
9575   rules~(with~the~subkeys~'color'~and~'width'),~
9576   sub-matrix~(several~subkeys)~
9577   and~xdots~(several~subkeys).~
9578   The~latter~is~for~the~command~\token_to_str:N \line.
9579 }
9580 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9581 {
9582   Unknown~key.\\
9583   The~key~'\l_keys_key_str'~is~unknown.\\
9584   It~will~be~ignored. \\
9585   \c_@@_available_keys_str
9586 }
9587 {
9588   The~available~keys~are~(in~alphabetic~order):~
9589   create-cell-nodes,~
9590   delimiters/color~and~
9591   sub-matrix~(several~subkeys).
9592 }
9593 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9594 {
9595   Unknown~key.\\
9596   The~key~'\l_keys_key_str'~is~unknown.\\
9597   That~key~will~be~ignored. \\
9598   \c_@@_available_keys_str
9599 }
9600 {
9601   The~available~keys~are~(in~alphabetic~order):~
9602   'delimiters/color',~
9603   'extra-height',~
9604   'hlines',~
9605   'hvlines',~
9606   'left-xshift',~
9607   'name',~
9608   'right-xshift',~
9609   'rules'~(with~the~subkeys~'color'~and~'width'),~
9610   'slim',~
9611   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9612   and~'right-xshift').\\
9613 }
9614 \@@_msg_new:nnn { Unknown~key~for~notes }
9615 {
9616   Unknown~key.\\
9617   The~key~'\l_keys_key_str'~is~unknown.\\
9618   That~key~will~be~ignored. \\
9619   \c_@@_available_keys_str
9620 }
9621 {
9622   The~available~keys~are~(in~alphabetic~order):~
9623   bottomrule,~
9624   code-after,~
9625   code-before,~
9626   detect-duplicates,~
9627   enumitem-keys,~
9628   enumitem-keys-para,~
9629   para,~
9630   label-in-list,~
9631   label-in-tabular~and~
9632   style.
9633 }

```

```

9634 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9635 {
9636   Unknown~key.\\
9637   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9638   \token_to_str:N \RowStyle. \\
9639   That~key~will~be~ignored. \\
9640   \c_@@_available_keys_str
9641 }
9642 {
9643   The~available~keys~are~(in~alphabetic~order):~
9644   'bold',~
9645   'cell-space-top-limit',~
9646   'cell-space-bottom-limit',~
9647   'cell-space-limits',~
9648   'color',~
9649   'nb-rows'~and~
9650   'rowcolor'.
9651 }
9652 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9653 {
9654   Unknown~key.\\
9655   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9656   \token_to_str:N \NiceMatrixOptions. \\
9657   That~key~will~be~ignored. \\
9658   \c_@@_available_keys_str
9659 }
9660 {
9661   The~available~keys~are~(in~alphabetic~order):~
9662   allow-duplicate-names,~
9663   caption-above,~
9664   cell-space-bottom-limit,~
9665   cell-space-limits,~
9666   cell-space-top-limit,~
9667   code-for-first-col,~
9668   code-for-first-row,~
9669   code-for-last-col,~
9670   code-for-last-row,~
9671   corners,~
9672   custom-key,~
9673   create-extra-nodes,~
9674   create-medium-nodes,~
9675   create-large-nodes,~
9676   delimiters~(several~subkeys),~
9677   end-of-row,~
9678   first-col,~
9679   first-row,~
9680   hlines,~
9681   hvlines,~
9682   hvlines-except-borders,~
9683   last-col,~
9684   last-row,~
9685   left-margin,~
9686   light-syntax,~
9687   matrix/columns-type,~
9688   notes~(several~subkeys),~
9689   nullify-dots,~
9690   pgf-node-code,~
9691   renew-dots,~
9692   renew-matrix,~
9693   respect-arraystretch,~
9694   rounded-corners,~
9695   right-margin,~
9696   rules~(with~the~subkeys~'color'~and~'width'),~

```

```

9697     small,~
9698     sub-matrix~(several~subkeys),~
9699     vlines,~
9700     xdots~(several~subkeys).
9701 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no 1 and r.

```

9702 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9703 {
9704     Unknown~key.\\
9705     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9706     \{NiceArray\}. \\
9707     That~key~will~be~ignored. \\
9708     \c_@@_available_keys_str
9709 }
9710 {
9711     The~available~keys~are~(in~alphabetic~order):~
9712     b,~
9713     baseline,~
9714     c,~
9715     cell-space-bottom-limit,~
9716     cell-space-limits,~
9717     cell-space-top-limit,~
9718     code-after,~
9719     code-for-first-col,~
9720     code-for-first-row,~
9721     code-for-last-col,~
9722     code-for-last-row,~
9723     color-inside,~
9724     columns-width,~
9725     corners,~
9726     create-extra-nodes,~
9727     create-medium-nodes,~
9728     create-large-nodes,~
9729     extra-left-margin,~
9730     extra-right-margin,~
9731     first-col,~
9732     first-row,~
9733     hlines,~
9734     hvlines,~
9735     hvlines-except-borders,~
9736     last-col,~
9737     last-row,~
9738     left-margin,~
9739     light-syntax,~
9740     name,~
9741     nullify-dots,~
9742     pgf-node-code,~
9743     renew-dots,~
9744     respect-arraystretch,~
9745     right-margin,~
9746     rounded-corners,~
9747     rules~(with~the~subkeys~'color'~and~'width'),~
9748     small,~
9749     t,~
9750     vlines,~
9751     xdots/color,~
9752     xdots/shorten-start,~
9753     xdots/shorten-end,~
9754     xdots/shorten~and~
9755     xdots/line-style.
9756 }

```

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).

```

9757 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9758 {
9759   Unknown~key.\\
9760   The~key~'\l_keys_key_str'~is~unknown~for~the~
9761   \@@_full_name_env:. \\
9762   That~key~will~be~ignored. \\
9763   \c_@@_available_keys_str
9764 }
9765 {
9766   The~available~keys~are~(in~alphabetic~order):~
9767   b,~
9768   baseline,~
9769   c,~
9770   cell-space-bottom-limit,~
9771   cell-space-limits,~
9772   cell-space-top-limit,~
9773   code-after,~
9774   code-for-first-col,~
9775   code-for-first-row,~
9776   code-for-last-col,~
9777   code-for-last-row,~
9778   color-inside,~
9779   columns-type,~
9780   columns-width,~
9781   corners,~
9782   create-extra-nodes,~
9783   create-medium-nodes,~
9784   create-large-nodes,~
9785   extra-left-margin,~
9786   extra-right-margin,~
9787   first-col,~
9788   first-row,~
9789   hlines,~
9790   hvlines,~
9791   hvlines-except-borders,~
9792   l,~
9793   last-col,~
9794   last-row,~
9795   left-margin,~
9796   light-syntax,~
9797   name,~
9798   nullify-dots,~
9799   pgf-node-code,~
9800   r,~
9801   renew-dots,~
9802   respect-arraystretch,~
9803   right-margin,~
9804   rounded-corners,~
9805   rules~(with~the~subkeys~'color'~and~'width'),~
9806   small,~
9807   t,~
9808   vlines,~
9809   xdots/color,~
9810   xdots/shorten-start,~
9811   xdots/shorten-end,~
9812   xdots/shorten~and~
9813   xdots/line-style.
9814 }
9815 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9816 {
9817   Unknown~key.\\

```



```

9818 The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9819 \{NiceTabular\}. \\
9820 That~key~will~be~ignored. \\
9821 \c_@@_available_keys_str
9822 }
9823 {
9824 The~available~keys~are~(in~alphabetic~order):~
9825 b,~
9826 baseline,~
9827 c,~
9828 caption,~
9829 cell-space-bottom-limit,~
9830 cell-space-limits,~
9831 cell-space-top-limit,~
9832 code-after,~
9833 code-for-first-col,~
9834 code-for-first-row,~
9835 code-for-last-col,~
9836 code-for-last-row,~
9837 color-inside,~
9838 columns-width,~
9839 corners,~
9840 custom-line,~
9841 create-extra-nodes,~
9842 create-medium-nodes,~
9843 create-large-nodes,~
9844 extra-left-margin,~
9845 extra-right-margin,~
9846 first-col,~
9847 first-row,~
9848 hlines,~
9849 hvlines,~
9850 hvlines-except-borders,~
9851 label,~
9852 last-col,~
9853 last-row,~
9854 left-margin,~
9855 light-syntax,~
9856 name,~
9857 notes~(several~subkeys),~
9858 nullify-dots,~
9859 pgf-node-code,~
9860 renew-dots,~
9861 respect-arraystretch,~
9862 right-margin,~
9863 rounded-corners,~
9864 rules~(with~the~subkeys~'color'~and~'width'),~
9865 short-caption,~
9866 t,~
9867 tabularnote,~
9868 vlines,~
9869 xdots/color,~
9870 xdots/shorten-start,~
9871 xdots/shorten-end,~
9872 xdots/shorten~and~
9873 xdots/line-style.
9874 }
9875 \@@_msg_new:nnn { Duplicate-name }
9876 {
9877 Duplicate-name.\\
9878 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9879 the~same~environment~name~twice.~You~can~go~on,~but,~
9880 maybe,~you~will~have~incorrect~results~especially~

```

```

9881   if~you~use~'columns-width=auto'~.~If~you~don't~want~to~see~this~
9882   message~again,~use~the~key~'allow-duplicate-names'~in~
9883   '\token_to_str:N \NiceMatrixOptions'.\\
9884   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9885   { For~a~list~of~the~names~already~used,~type~H~<return>. }
9886   }
9887   {
9888     The~names~already~defined~in~this~document~are:~
9889     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9890   }

9891 \@@_msg_new:nn { Option~auto~for~columns-width }
9892 {
9893   Erroneous~use.\\
9894   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9895   That~key~will~be~ignored.
9896 }

9897 \@@_msg_new:nn { NiceTabularX~without~X }
9898 {
9899   NiceTabularX~without~X.\\
9900   You~should~not~use~{NiceTabularX}~without~X~columns.\\
9901   However,~you~can~go~on.
9902 }

9903 \@@_msg_new:nn { Preamble~forgotten }
9904 {
9905   Preamble~forgotten.\\
9906   You~have~probably~forgotten~the~preamble~of~your~
9907   \@@_full_name_env:. \\
9908   This~error~is~fatal.
9909 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
3	Collecting options	4
4	Technical definitions	4
5	Parameters	10
6	The command <code>\tabularnote</code>	20
7	Command for creation of rectangle nodes	25
8	The options	26
9	Important code used by <code>{NiceArrayWithDelims}</code>	36
10	The <code>\CodeBefore</code>	49
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	57
13	The redefinition of <code>\multicolumn</code>	72
14	The environment <code>{NiceMatrix}</code> and its variants	89
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	90
16	After the construction of the array	91
17	We draw the dotted lines	98
18	The actual instructions for drawing the dotted lines with Tikz	111
19	User commands available in the new environments	116
20	The command <code>\line</code> accessible in code-after	123
21	The command <code>\RowStyle</code>	124
22	Colors of cells, rows and columns	126
23	The vertical and horizontal rules	137
24	The key corners	151
25	The environment <code>{NiceMatrixBlock}</code>	154
26	The extra nodes	155
27	The blocks	159
28	How to draw the dotted lines transparently	179
29	Automatic arrays	179
30	The redefinition of the command <code>\dotfill</code>	181

31	The command <code>\diagbox</code>	181
32	The keyword <code>\CodeAfter</code>	183
33	The delimiters in the preamble	183
34	The command <code>\SubMatrix</code>	185
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	193
36	The command <code>TikzEveryCell</code>	196
37	The command <code>\ShowCellNames</code>	197
38	We process the options at package loading	200
39	About the package underscore	201
40	Error messages of the package	202