

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

April 23, 2024

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`).

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.

```
9 \RequirePackage { amsmath }
```

```
10 \RequirePackage { array }
```

In the version 2.6a of `array`, important modifications have been done for the Tagging Project.

```
11 \bool_const:Nn \c_@@_tagging_array_bool { \cs_if_exist_p:N \ar@ialign }

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

*This document corresponds to the version 6.27b of `nicematrix`, at the date of 2024/04/23.

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).

```

20 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
21 {
22   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
23     { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
24     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
25 }

```

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

```

26 \cs_new_protected:Npn \@@_error_or_warning:n
27 { \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”.

```

28 \bool_new:N \g_@@_messages_for_Overleaf_bool
29 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
30 {
31   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
32   || \str_if_eq_p:on \c_sys_jobname_str { output } % for Overleaf
33 }

```

```

34 \cs_new_protected:Npn \@@_msg_redirect_name:nn
35 { \msg_redirect_name:nnn { nicematrix } }
36 \cs_new_protected:Npn \@@_gredirect_none:n #1
37 {
38   \group_begin:
39   \globaldefs = 1
40   \@@_msg_redirect_name:nn { #1 } { none }
41   \group_end:
42 }
43 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
44 {
45   \@@_error:n { #1 }
46   \@@_gredirect_none:n { #1 }
47 }
48 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
49 {
50   \@@_warning:n { #1 }
51   \@@_gredirect_none:n { #1 }
52 }

```

We will delete in the future the following lines which are only a security.

```

53 \cs_set:Npn \int_if_zero:NT #1 { \int_compare:nNnT #1 = \c_zero_int }
54 \cs_set:Npn \int_if_zero:NTF #1 { \int_compare:nNnTF #1 = \c_zero_int }

```

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.

```

55 \@@_msg_new:nn { Internal~error }
56 {
57   Potential~problem~when~using~nicematrix.\\
58   The~package~nicematrix~have~detected~a~modification~of~the~
59   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
60   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
61   this~message~again,~load~nicematrix~with:~\token_to_str:N
62   \usepackage[no-test-for-array]{nicematrix}.
63 }

64 \@@_msg_new:nn { mdwtab-loaded }
65 {
66   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
67   This~error~is~fatal.
68 }

69 \cs_new_protected:Npn \@@_security_test:n #1
70 {
71   \peek_meaning:NTF \ignorespaces
72   { \@@_security_test_i:w }
73   { \@@_error:n { Internal~error } }
74   #1
75 }

76 \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
77 {
78   \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
79   #1
80 }

```

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).

```

81 \hook_gput_code:nnn { begindocument / after } { . }
82 {
83   \IfPackageLoadedTF { mdwtab }
84   { \@@_fatal:n { mdwtab-loaded } }
85   {
86     \bool_if:NF \g_@@_no_test_for_array_bool
87     {
88       \group_begin:
89       \hbox_set:Nn \l_tmpa_box
90       {
91         \begin { tabular } { c > { \@@_security_test:n } c c }
92         text & & text
93         \end { tabular }
94       }
95       \group_end:
96     }
97   }
98 }

```

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`).

```

99 \cs_new_protected:Npn \@@_collect_options:n #1
100 {
101   \peek_meaning:NTF [
102     { \@@_collect_options:nw { #1 } }
103     { #1 { } }
104   }

```

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

```

105 \NewDocumentCommand \@@_collect_options:nw { m r[] }
106 { \@@_collect_options:nn { #1 } { #2 } }
107
108 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
109 {
110   \peek_meaning:NTF [
111     { \@@_collect_options:nnw { #1 } { #2 } }
112     { #1 { #2 } }
113   }
114
115 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
116 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

4 Technical definitions

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

```

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

```

The following token list will be used for definitions of user commands (with `\NewDocumentCommand`) with an embellishment using an *underscore* (there may be problems because of the catcode of the underscore).

```

133 \tl_new:N \l_@@_argspec_tl
134 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
135 \cs_generate_variant:Nn \str_lowercase:n { V }

136 \hook_gput_code:nnn { begindocument } { . }
137 {
138   \IfPackageLoadedTF { tikz }
139   {

```

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 `\hook_gput_code:nnn { begindocument } { . }` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

140   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
141   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
142 }
143 {
144   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
145   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
146 }
147 }

```

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 April 2024, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

148 \IfClassLoadedTF { revtex4-1 }
149 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
150 {
151   \IfClassLoadedTF { revtex4-2 }
152   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
153   {

```

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

```

154   \cs_if_exist:NT \rvtx@ifformat@geq
155   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
156   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
157 }
158 }

```

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.

```

159 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
160 {
161   \iow_now:Nn \@mainaux
162   {
163     \ExplSyntaxOn
164     \cs_if_free:NT \pgfsyspdfmark
165     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
166     \ExplSyntaxOff
167   }
168   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
169 }

```

We define a command `\iddots` similar to `\ddots` (``) but with dots going forward (``). 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.

```

170 \ProvideDocumentCommand \iddots { }
171 {
172   \mathinner
173   {
174     \tex_mkern:D 1 mu
175     \box_move_up:nn { 1 pt } { \hbox { . } }
176     \tex_mkern:D 2 mu
177     \box_move_up:nn { 4 pt } { \hbox { . } }
178     \tex_mkern:D 2 mu
179     \box_move_up:nn { 7 pt }
180     { \vbox:n { \kern 7 pt \hbox { . } } }
181     \tex_mkern:D 1 mu
182   }
183 }

```

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.

```

184 \hook_gput_code:nnn { begindocument } { . }
185 {
186   \IfPackageLoadedTF { booktabs }
187   { \iow_now:Nn \mainaux \nicematrix@redefine@check@rerun }
188   { }
189 }
190 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
191 {
192   \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`).

```

193   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
194   {
195     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
196     { \@@_old_pgful@check@rerun { ##1 } { ##2 } }
197   }
198 }

```

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

```

199 \hook_gput_code:nnn { begindocument } { . }
200 {
201   \IfPackageLoadedTF { colortbl }
202   { }
203   {

```

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.

```

204     \cs_set_protected:Npn \CT@arc@ { }
205     \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
206     \cs_set:Npn \CT@arc@ #1 #2
207     {
208       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
209       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
210     }

```

Idem for \CT@drs@.

```

211 \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
212 \cs_set:Npn \CT@drs #1 #2
213 {
214   \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
215   { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
216 }
217 \cs_set:Npn \hline
218 {
219   \noalign { \ifnum 0 = ' } \fi
220   \cs_set_eq:NN \hskip \vskip
221   \cs_set_eq:NN \vrule \hrule
222   \cs_set_eq:NN \@width \@height
223   { \CT@arc@ \vline }
224   \futurelet \reserved@a
225   \@xhline
226 }
227 }
228 }

```

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.

```

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

```

The following \skip_horizontal:N \c_zero_dim is to prevent a potential \unskip to delete the \leaders¹

```

239 \skip_horizontal:N \c_zero_dim
240 }

```

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.

```

241 \everycr { }
242 \cr
243 \noalign { \skip_vertical:N -\arrayrulewidth }
244 }

```

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.

```

245 \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.

```

246 { \@@_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*.

```

247 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
248 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
249 {

```

¹See question 99041 on TeX StackExchange.

```

250 \tl_if_empty:nTF { #3 }
251 { \@@_cline_iii:w #1|#2-#2 \q_stop }
252 { \@@_cline_ii:w #1|#2-#3 \q_stop }
253 }
254 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
255 { \@@_cline_iii:w #1|#2-#3 \q_stop }
256 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
257 {

```

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).

```

258 \int_compare:nNt { #1 } < { #2 }
259 { \multispan { \int_eval:n { #2 - #1 } } & }
260 \multispan { \int_eval:n { #3 - #2 + 1 } }
261 {
262 \CT@arc@
263 \leaders \hrule \@height \arrayrulewidth \hfill
264 \skip_horizontal:N \c_zero_dim
265 }

```

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

```

266 \peek_meaning_remove_ignore_spaces:NTF \cline
267 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
268 { \everycr { } \cr }
269 }
270 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

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

```

271 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

```

```

272 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
273 {
274 \tl_if_blank:nF { #1 }
275 {
276 \tl_if_head_eq_meaning:nNTF { #1 } [
277 { \cs_set:Npn \CT@arc@ { \color #1 } }
278 { \cs_set:Npn \CT@arc@ { \color { #1 } } }
279 ]
280 }
281 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }

```

```

282 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
283 {
284 \tl_if_head_eq_meaning:nNTF { #1 } [
285 { \cs_set:Npn \CT@drsc@ { \color #1 } }
286 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
287 ]
288 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }

```

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

```

289 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
290 {
291 \tl_if_head_eq_meaning:nNTF { #2 } [
292 { #1 #2 }
293 { #1 { #2 } }
294 ]
295 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```


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

```

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

299 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

300 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
301 {
302   \tl_set_rescan:Nno
303     #1
304     {
305       \char_set_catcode_other:N >
306       \char_set_catcode_other:N <
307     }
308     #1
309 }

```

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.

```

310 \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).

```

311 \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.

```

312 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
313 { \int_use:N \g_@@_env_int }

```

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

```

314 \cs_new_protected:Npn \@@_qpoint:n #1
315 { \pgfpaintanchor { \@@_env: - #1 } { center } }

```

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

```

316 \bool_new:N \l_@@_tabular_bool

```

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

```

317 \bool_new:N \g_@@_delims_bool
318 \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.

```
319 \bool_new:N \l_@@_preamble_bool
320 \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.

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

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

```
322 \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.

```
323 \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).

```
324 \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.).

```
325 \dim_new:N \l_@@_col_width_dim
326 \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.

```
327 \int_new:N \g_@@_row_total_int
328 \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).

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

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

```
330 \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[l]{3cm}` will provide the value `l` for all the cells of the column.

```
331 \tl_new:N \l_@@_hpos_cell_tl
332 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
334 \dim_new:N \g_@@_blocks_ht_dim
335 \dim_new:N \g_@@_blocks_dp_dim
```

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

```
336 \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`.

```
337 \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.

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

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

```
339 \bool_new:N \l_@@_notes_detect_duplicates_bool
340 \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.

```
341 \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).

```
342 \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).

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

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

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

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

```
345 \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.

```
346 \bool_new:N \l_@@_X_bool
347 \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 }`).

```
348 \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.

```
349 \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.

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

351 \tl_new:N \g_@@_left_delim_tl
352 \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}`).

```
353 \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`).

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

For `\multicolumn`.

```
355 \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`.

```
356 \tl_new:N \l_@@_columns_type_tl
357 \str_set:Nn \l_@@_columns_type_tl { c }
```

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

```
358 \tl_new:N \l_@@_xdots_down_tl
359 \tl_new:N \l_@@_xdots_up_tl
360 \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`).

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

362 \cs_new_protected:Npn \@@_test_if_math_mode:
363 {
364   \if_mode_math: \else:
365     \@@_fatal:n { Outside-math-mode }
366   \fi:
367 }
```

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.

```
368 \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”.

```
369 \colorlet { nicematrix-last-col } { . }
370 \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`).

```
371 \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*.

```
372 \tl_new:N \g_@@_com_or_env_str
373 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
374 \bool_new:N \l_@@_bold_row_style_bool
```

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).

```
375 \cs_new:Npn \@@_full_name_env:
376 {
377   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
378     { command \space \c_backslash_str \g_@@_name_env_str }
379     { environment \space \{ \g_@@_name_env_str \} }
380 }
```

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

```
381 \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.

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

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

```
383 \tl_new:N \g_@@_pre_code_before_tl
384 \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.

```
385 \tl_new:N \g_@@_pre_code_after_tl
386 \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.

```
387 \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.

```
388 \int_new:N \l_@@_old_iRow_int
389 \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).

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

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

```
391 \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.

```
392 \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.

```
393 \bool_new:N \l_@@_X_columns_aux_bool
394 \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.

```
395 \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).

```
396 \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).

```
397 \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`).

```
398 \tl_new:N \l_@@_code_before_tl
399 \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).

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

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

```
401 \dim_new:N \l_@@_x_initial_dim
402 \dim_new:N \l_@@_y_initial_dim
403 \dim_new:N \l_@@_x_final_dim
404 \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.

```
405 \dim_zero_new:N \l_@@_tmpc_dim
406 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
407 \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”.

```
408 \dim_new:N \g_@@_width_last_col_dim
409 \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.

```
410 \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!).

```
411 \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}`.

```
412 \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.

```
413 \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.

```
414 \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`).

```
415 \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.

```
416 \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).

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

```
418 \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`).

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

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

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

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

The following counters will be used when drawing the rules.

```

423 \int_new:N \l_@@_start_int
424 \int_set_eq:NN \l_@@_start_int \c_one_int
425 \int_new:N \l_@@_end_int
426 \int_new:N \l_@@_local_start_int
427 \int_new:N \l_@@_local_end_int

```

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

```

428 \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).

```

429 \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`.

```

430 \tl_new:N \l_@@_fill_tl
431 \tl_new:N \l_@@_opacity_tl
432 \tl_new:N \l_@@_draw_tl
433 \seq_new:N \l_@@_tikz_seq
434 \clist_new:N \l_@@_borders_clist
435 \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.

```

436 \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`.

```

437 \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.

```

438 \dim_new:N \l_@@_offset_dim

```

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

```

439 \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`).

```

440 \str_new:N \l_@@_hpos_block_str
441 \str_set:Nn \l_@@_hpos_block_str { c }
442 \bool_new:N \l_@@_hpos_of_block_cap_bool

```

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

```

443 \bool_new:N \l_@@_nocolor_used_bool

```

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

```

444 \str_new:N \l_@@_vpos_block_str
445 \str_set:Nn \l_@@_vpos_block_str { c }

```


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

```
446 \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).

```
447 \bool_new:N \l_@@_vlines_block_bool
```

```
448 \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.

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

```
450 \dim_new:N \l_@@_submatrix_extra_height_dim
```

```
451 \dim_new:N \l_@@_submatrix_left_xshift_dim
```

```
452 \dim_new:N \l_@@_submatrix_right_xshift_dim
```

```
453 \clist_new:N \l_@@_hlines_clist
```

```
454 \clist_new:N \l_@@_vlines_clist
```

```
455 \clist_new:N \l_@@_submatrix_hlines_clist
```

```
456 \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}`).

```
457 \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.

```
458 \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`).

```
459 \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.

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

```
461 \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.

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

```
463 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

• Last row

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

```
464 \int_new:N \l_@@_last_row_int
```

```
465 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
467 \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`.

```
468 \int_new:N \l_@@_last_col_int
469 \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.

```
470 \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`).

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

Some utilities

```
472 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
473 {
474   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
475   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
476 }
```

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.

```
477 \cs_new_protected:Npn \@@_expand_clist:N #1
478 {
479   \clist_if_in:NVF #1 \c_@@_all_tl
480   {
481     \clist_clear:N \l_tmpa_clist
482     \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.

```

483     {
484       \tl_if_in:nnTF { ##1 } { - }
485       { \@@_cut_on_hyphen:w ##1 \q_stop }
486       {
487         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
488         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
489       }
490       \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
491       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
492     }
493     \tl_set_eq:NN #1 \l_tmpa_clist
494   }
495 }

```

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.

```

496 \hook_gput_code:nnn { begindocument } { . }
497 {
498   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
499   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
500   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
501 }

```

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 component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component 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 expressed by `\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 components 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.

```

502 \newcounter { tabularnote }
503 \seq_new:N \g_@@_notes_seq
504 \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.

```

505 \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.

```

506 \seq_new:N \l_@@_notes_labels_seq
507 \newcounter{nicematrix_draft}
508 \cs_new_protected:Npn \@@_notes_format:n #1
509 {
510   \setcounter { nicematrix_draft } { #1 }
511   \@@_notes_style:n { nicematrix_draft }
512 }

```

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

```

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

```

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

```

514 \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`.

```

515 \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`.

```

516 \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`).

```

517 \hook_gput_code:nnn { begindocument } { . }
518 {
519   \IfPackageLoadedTF { enumitem }
520   {

```

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.

```

521     \newlist { tabularnotes } { enumerate } { 1 }
522     \setlist [ tabularnotes ]
523     {
524         topsep = Opt ,
525         noitemsep ,
526         leftmargin = * ,
527         align = left ,
528         labelsep = Opt ,
529         label =
530             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
531     }
532     \newlist { tabularnotes* } { enumerate* } { 1 }
533     \setlist [ tabularnotes* ]
534     {
535         afterlabel = \nobreak ,
536         itemjoin = \quad ,
537         label =
538             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
539     }

```

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`.

```

540     \NewDocumentCommand \tabularnote { o m }
541     {
542         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capytype } \l_@@_in_env_bool
543         {
544             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
545             { \@@_error:n { tabularnote~forbidden } }
546             {
547                 \bool_if:NTF \l_@@_in_caption_bool
548                 \@@_tabularnote_caption:nn
549                 \@@_tabularnote:nn
550                 { #1 } { #2 }
551             }
552         }
553     }
554 }
555 {
556     \NewDocumentCommand \tabularnote { o m }
557     {
558         \@@_error_or_warning:n { enumitem~not~loaded }
559         \@@_gredirect_none:n { enumitem~not~loaded }
560     }
561 }
562 }

563 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
564 { \tl_if_novalue:nT { #1 } { #3 } }

```

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 expressed by `\c_novalue_tl`) and `#2` is the mandatory argument of `\tabularnote`.

```

565 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
566 {

```

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).

```

567     \int_zero:N \l_tmpa_int
568     \bool_if:NT \l_@@_notes_detect_duplicates_bool
569     {

```

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

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

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

When we will go through the sequence `\g_@@_notes_seq`, we will count in `\l_tmpb_int` the notes without explicit label in order to have the “current” value of the counter `\c@tabulernote`.

```

570     \int_zero:N \l_tmpb_int
571     \seq_map_indexed_inline:Nn \g_@@_notes_seq
572     {
573         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
574         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
575             {
576                 \tl_if_novalue:nTF { #1 }
577                     { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
578                     { \int_set:Nn \l_tmpa_int { ##1 } }
579                 \seq_map_break:
580             }
581         }
582         \int_if_zero:nF \l_tmpa_int
583         { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
584     }
585     \int_if_zero:nT \l_tmpa_int
586     {
587         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
588         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
589     }
590     \seq_put_right:Nx \l_@@_notes_labels_seq
591     {
592         \tl_if_novalue:nTF { #1 }
593         {
594             \@@_notes_format:n
595             {
596                 \int_eval:n
597                 {
598                     \int_if_zero:nTF \l_tmpa_int
599                     \c@tabulernote
600                     \l_tmpa_int
601                 }
602             }
603         }
604         { #1 }
605     }
606     \peek_meaning:NF \tabulernote
607     {

```

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

```

608     \hbox_set:Nn \l_tmpa_box
609     {

```

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

```

610         \@@_notes_label_in_tabular:n
611         {

```

```

612         \seq_use:Nnnn
613         \l_@@_notes_labels_seq { , } { , } { , }
614     }
615 }

```

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

```

616     \int_gdecr:N \c@tabularnote
617     \int_set_eq:NN \l_tmpa_int \c@tabularnote
618     \refstepcounter { tabularnote }
619     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
620     { \int_gincr:N \c@tabularnote }
621     \seq_clear:N \l_@@_notes_labels_seq
622     \bool_lazy_or:nnTF
623     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
624     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
625     {
626         \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).

```

627         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
628     }
629     { \box_use:N \l_tmpa_box }
630 }
631 }

```

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`.

```

632 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
633 {
634     \bool_if:NTF \g_@@_caption_finished_bool
635     {
636         \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
637         { \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`!

```

638     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
639     { \@@_error:n { Identical-notes-in-caption } }
640 }
641 {

```

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

```

642     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
643     {

```

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.

```

644         \bool_gset_true:N \g_@@_caption_finished_bool
645         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
646         \int_gzero:N \c@tabularnote
647     }
648     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
649 }

```

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!

```

650 \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
651 \seq_put_right:Nx \l_@@_notes_labels_seq
652 {
653   \tl_if_novalue:nTF { #1 }
654     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
655     { #1 }
656 }
657 \peek_meaning:NF \tabularnote
658 {
659   \@@_notes_label_in_tabular:n
660     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
661   \seq_clear:N \l_@@_notes_labels_seq
662 }
663 }

664 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
665 { \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.

```

666 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
667 {
668   \begin { pgfscope }
669   \pgfset
670   {
671     inner~sep = \c_zero_dim ,
672     minimum~size = \c_zero_dim
673   }
674   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
675   \pgfnode
676   { rectangle }
677   { center }
678   {
679     \vbox_to_ht:nn
680     { \dim_abs:n { #5 - #3 } }
681     {
682       \vfill
683       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
684     }
685   }
686   { #1 }
687   { }
688   \end { pgfscope }
689 }

```

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.

```

690 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
691 {
692   \begin { pgfscope }
693   \pgfset
694   {
695     inner~sep = \c_zero_dim ,
696     minimum~size = \c_zero_dim

```



```

697     }
698     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
699     \pgfpointdiff { #3 } { #2 }
700     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
701     \pgfnode
702     { rectangle }
703     { center }
704     {
705         \vbox_to_ht:nn
706         { \dim_abs:n \l_tmpb_dim }
707         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
708     }
709     { #1 }
710     { }
711     \end { pgfscope }
712 }

```

8 The options

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

```

713 \tl_new:N \l_@@_caption_tl
714 \tl_new:N \l_@@_short_caption_tl
715 \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).

```

716 \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.

```

717 \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`.

```

718 \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`).

```

719 \dim_new:N \l_@@_cell_space_top_limit_dim
720 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

721 \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.

```

722 \dim_new:N \l_@@_xdots_inter_dim
723 \hook_gput_code:nnn { begindocument } { . }
724 { \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).

```

725 \dim_new:N \l_@@_xdots_shorten_start_dim
726 \dim_new:N \l_@@_xdots_shorten_end_dim
727 \hook_gput_code:nnn { begindocument } { . }
728 {
729   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
730   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
731 }

```

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.

```

732 \dim_new:N \l_@@_xdots_radius_dim
733 \hook_gput_code:nnn { begindocument } { . }
734 { \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.

```

735 \tl_new:N \l_@@_xdots_line_style_tl
736 \tl_const:Nn \c_@@_standard_tl { standard }
737 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax` and the boolean `\l_@@_light_syntax_expanded_bool` correspond to the the option `light-syntax-expanded`.

```

738 \bool_new:N \l_@@_light_syntax_bool
739 \bool_new:N \l_@@_light_syntax_expanded_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).

```

740 \tl_new:N \l_@@_baseline_tl
741 \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`).

```

742 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

743 \bool_new:N \l_@@_parallelize_diags_bool
744 \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.

```

745 \clist_new:N \l_@@_corners_clist

```

```

746 \dim_new:N \l_@@_notes_above_space_dim
747 \hook_gput_code:nnn { begindocument } { . }
748 { \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.).

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

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

```
750 \cs_new_protected:Npn \@@_reset_arraystretch:
751 { \cs_set_nopar:Npn \arraystretch { 1 } }
```

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

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

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

```
753 \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.

```
754 \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”.

```
755 \bool_new:N \l_@@_medium_nodes_bool
756 \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).

```
757 \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`).

```
758 \dim_new:N \l_@@_left_margin_dim
759 \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`.

```
760 \dim_new:N \l_@@_extra_left_margin_dim
761 \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.

```
762 \tl_new:N \l_@@_end_of_row_tl
763 \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 “:”.

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

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

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

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

```

766 \bool_new:N \l_@@_delimiters_max_width_bool

767 \keys_define:nn { NiceMatrix / xdots }
768 {
769   shorten-start .code:n =
770     \hook_gput_code:nnn { begindocument } { . }
771     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
772   shorten-end .code:n =
773     \hook_gput_code:nnn { begindocument } { . }
774     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
775   shorten-start .value_required:n = true ,
776   shorten-end .value_required:n = true ,
777   shorten .code:n =
778     \hook_gput_code:nnn { begindocument } { . }
779     {
780       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
781       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
782     } ,
783   shorten .value_required:n = true ,
784   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
785   horizontal-labels .default:n = true ,
786   line-style .code:n =
787     {
788       \bool_lazy_or:nnTF
789         { \cs_if_exist_p:N \tikzpicture }
790         { \str_if_eq_p:nn { #1 } { standard } }
791         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
792         { @@_error:n { bad-option-for~line-style } }
793     } ,
794   line-style .value_required:n = true ,
795   color .tl_set:N = \l_@@_xdots_color_tl ,
796   color .value_required:n = true ,
797   radius .code:n =
798     \hook_gput_code:nnn { begindocument } { . }
799     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
800   radius .value_required:n = true ,
801   inter .code:n =
802     \hook_gput_code:nnn { begindocument } { . }
803     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
804   radius .value_required:n = true ,

```

The options `down`, `up` and `middle` are not documented for the final user because he should use the syntax with `^`, `_` and `::`. We use `\tl_put_right:Nn` and not `\tl_set:Nn` (or `.tl_set:N`) because we don't want a direct use of `up=...` erased by an absent `^{\dots}`.

```

805   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
806   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
807   middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

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

```

808   draw-first .code:n = \prg_do_nothing: ,
809   unknown .code:n = @@_error:n { Unknown~key~for~xdots }
810 }

```

```

811 \keys_define:nn { NiceMatrix / rules }
812 {
813   color .tl_set:N = \l_@@_rules_color_tl ,
814   color .value_required:n = true ,
815   width .dim_set:N = \arrayrulewidth ,
816   width .value_required:n = true ,
817   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
818 }

```

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

```

819 \keys_define:nn { NiceMatrix / Global }
820 {
821   no-cell-nodes .code:n =
822     \cs_set_protected:Npn \@@_node_for_cell:
823     { \box_use_drop:N \l_@@_cell_box } ,
824   no-cell-nodes .value_forbidden:n = true ,
825   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
826   rounded-corners .default:n = 4 pt ,
827   custom-line .code:n = \@@_custom_line:n { #1 } ,
828   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
829   rules .value_required:n = true ,
830   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
831   standard-cline .default:n = true ,
832   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
833   cell-space-top-limit .value_required:n = true ,
834   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
835   cell-space-bottom-limit .value_required:n = true ,
836   cell-space-limits .meta:n =
837   {
838     cell-space-top-limit = #1 ,
839     cell-space-bottom-limit = #1 ,
840   } ,
841   cell-space-limits .value_required:n = true ,
842   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
843   light-syntax .code:n =
844     \bool_set_true:N \l_@@_light_syntax_bool
845     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
846   light-syntax .value_forbidden:n = true ,
847   light-syntax-expanded .code:n =
848     \bool_set_true:N \l_@@_light_syntax_bool
849     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
850   light-syntax-expanded .value_forbidden:n = true ,
851   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
852   end-of-row .value_required:n = true ,
853   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
854   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
855   last-row .int_set:N = \l_@@_last_row_int ,
856   last-row .default:n = -1 ,
857   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
858   code-for-first-col .value_required:n = true ,
859   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
860   code-for-last-col .value_required:n = true ,
861   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
862   code-for-first-row .value_required:n = true ,
863   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
864   code-for-last-row .value_required:n = true ,
865   hlines .clist_set:N = \l_@@_hlines_clist ,
866   vlines .clist_set:N = \l_@@_vlines_clist ,
867   hlines .default:n = all ,
868   vlines .default:n = all ,
869   vlines-in-sub-matrix .code:n =
870   {

```

```

871 \tl_if_single_token:nTF { #1 }
872 {
873     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
874     { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

875     { \cs_set_eq:cN { @@_#1 } \@@_make_preamble_vlism:n }
876 }
877 { \@@_error:n { One~letter~allowed } }
878 } ,
879 vlines-in-sub-matrix .value_required:n = true ,
880 hvlines .code:n =
881 {
882     \bool_set_true:N \l_@@_hvlines_bool
883     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
884     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
885 } ,
886 hvlines-except-borders .code:n =
887 {
888     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
889     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
890     \bool_set_true:N \l_@@_hvlines_bool
891     \bool_set_true:N \l_@@_except_borders_bool
892 } ,
893 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.

```

894 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
895 renew-dots .value_forbidden:n = true ,
896 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
897 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
898 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
899 create-extra-nodes .meta:n =
900 { create-medium-nodes , create-large-nodes } ,
901 left-margin .dim_set:N = \l_@@_left_margin_dim ,
902 left-margin .default:n = \arraycolsep ,
903 right-margin .dim_set:N = \l_@@_right_margin_dim ,
904 right-margin .default:n = \arraycolsep ,
905 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
906 margin .default:n = \arraycolsep ,
907 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
908 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
909 extra-margin .meta:n =
910 { extra-left-margin = #1 , extra-right-margin = #1 } ,
911 extra-margin .value_required:n = true ,
912 respect-arraystretch .code:n =
913 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
914 respect-arraystretch .value_forbidden:n = true ,
915 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
916 pgf-node-code .value_required:n = true
917 }

```

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

```

918 \keys_define:nn { NiceMatrix / Env }
919 {
920     corners .clist_set:N = \l_@@_corners_clist ,
921     corners .default:n = { NW , SW , NE , SE } ,
922     code-before .code:n =
923     {
924         \tl_if_empty:nF { #1 }

```

```

925     {
926       \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
927       \bool_set_true:N \l_@@_code_before_bool
928     }
929   } ,
930   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}`.

```

931   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
932   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
933   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
934   baseline .tl_set:N = \l_@@_baseline_tl ,
935   baseline .value_required:n = true ,
936   columns-width .code:n =
937     \tl_if_eq:nnTF { #1 } { auto }
938     { \bool_set_true:N \l_@@_auto_columns_width_bool }
939     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
940   columns-width .value_required:n = true ,
941   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.

```

942   \legacy_if:nF { measuring@ }
943   {
944     \str_set:Nx \l_tmpa_str { #1 }
945     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
946     { \@@_error:nn { Duplicate~name } { #1 } }
947     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
948     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
949   } ,
950   name .value_required:n = true ,
951   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
952   code-after .value_required:n = true ,
953   color-inside .code:n =
954     \bool_set_true:N \l_@@_color_inside_bool
955     \bool_set_true:N \l_@@_code_before_bool ,
956   color-inside .value_forbidden:n = true ,
957   colortbl-like .meta:n = color-inside
958 }
959 \keys_define:nn { NiceMatrix / notes }
960 {
961   para .bool_set:N = \l_@@_notes_para_bool ,
962   para .default:n = true ,
963   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
964   code-before .value_required:n = true ,
965   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
966   code-after .value_required:n = true ,
967   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
968   bottomrule .default:n = true ,
969   style .cs_set:Np = \@@_notes_style:n #1 ,
970   style .value_required:n = true ,
971   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
972   label-in-tabular .value_required:n = true ,
973   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
974   label-in-list .value_required:n = true ,
975   enumitem-keys .code:n =
976     {
977       \hook_gput_code:nnn { begindocument } { . }
978       {
979         \IfPackageLoadedTF { enumitem }
980         { \setlist* [ tabularnotes ] { #1 } }
981         { }

```

```

982     }
983   } ,
984   enumitem-keys .value_required:n = true ,
985   enumitem-keys-para .code:n =
986   {
987     \hook_gput_code:nnn { begindocument } { . }
988     {
989       \IfPackageLoadedTF { enumitem }
990       { \setlist* [ tabularnotes* ] { #1 } }
991       { }
992     }
993   } ,
994   enumitem-keys-para .value_required:n = true ,
995   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
996   detect-duplicates .default:n = true ,
997   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
998 }
999 \keys_define:nn { NiceMatrix / delimiters }
1000 {
1001   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1002   max-width .default:n = true ,
1003   color .tl_set:N = \l_@@_delimiters_color_tl ,
1004   color .value_required:n = true ,
1005 }

```

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

```

1006 \keys_define:nn { NiceMatrix }
1007 {
1008   NiceMatrixOptions .inherit:n =
1009   { NiceMatrix / Global } ,
1010   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1011   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1012   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1013   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1014   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1015   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1016   CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1017   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1018   NiceMatrix .inherit:n =
1019   {
1020     NiceMatrix / Global ,
1021     NiceMatrix / Env ,
1022   } ,
1023   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1024   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1025   NiceTabular .inherit:n =
1026   {
1027     NiceMatrix / Global ,
1028     NiceMatrix / Env
1029   } ,
1030   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1031   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1032   NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1033   NiceArray .inherit:n =
1034   {
1035     NiceMatrix / Global ,
1036     NiceMatrix / Env ,
1037   } ,
1038   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1039   NiceArray / rules .inherit:n = NiceMatrix / rules ,
1040   pNiceArray .inherit:n =

```



```

1041     {
1042       NiceMatrix / Global ,
1043       NiceMatrix / Env ,
1044     } ,
1045     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1046     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1047   }

```

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

```

1048 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1049 {
1050   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1051   delimiters / color .value_required:n = true ,
1052   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1053   delimiters / max-width .default:n = true ,
1054   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1055   delimiters .value_required:n = true ,
1056   width .dim_set:N = \l_@@_width_dim ,
1057   width .value_required:n = true ,
1058   last-col .code:n =
1059     \tl_if_empty:nF { #1 }
1060     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1061     \int_zero:N \l_@@_last_col_int ,
1062   small .bool_set:N = \l_@@_small_bool ,
1063   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.

```

1064   renew-matrix .code:n = \@@_renew_matrix: ,
1065   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}`.

```

1066   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.

```

1067   columns-width .code:n =
1068     \tl_if_eq:nnTF { #1 } { auto }
1069     { \@@_error:n { Option~auto~for~columns-width } }
1070     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1071   allow-duplicate-names .code:n =
1072     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1073   allow-duplicate-names .value_forbidden:n = true ,
1074   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1075   notes .value_required:n = true ,
1076   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1077   sub-matrix .value_required:n = true ,
1078   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1079   matrix / columns-type .value_required:n = true ,
1080   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1081   caption-above .default:n = true ,
1082   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1083 }

```

`\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.

```
1084 \NewDocumentCommand \NiceMatrixOptions { m }
1085 { \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.

```
1086 \keys_define:nn { NiceMatrix / NiceMatrix }
1087 {
1088   last-col .code:n = \tl_if_empty:nTF { #1 }
1089               {
1090                 \bool_set_true:N \l_@@_last_col_without_value_bool
1091                 \int_set:Nn \l_@@_last_col_int { -1 }
1092               }
1093               { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1094   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1095   columns-type .value_required:n = true ,
1096   l .meta:n = { columns-type = l } ,
1097   r .meta:n = { columns-type = r } ,
1098   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1099   delimiters / color .value_required:n = true ,
1100   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1101   delimiters / max-width .default:n = true ,
1102   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1103   delimiters .value_required:n = true ,
1104   small .bool_set:N = \l_@@_small_bool ,
1105   small .value_forbidden:n = true ,
1106   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1107 }
```

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

```
1108 \keys_define:nn { NiceMatrix / NiceArray }
1109 {
```

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.

```
1110   small .bool_set:N = \l_@@_small_bool ,
1111   small .value_forbidden:n = true ,
1112   last-col .code:n = \tl_if_empty:nF { #1 }
1113                       { \@@_error:n { last-col-non-empty-for-NiceArray } }
1114                       \int_zero:N \l_@@_last_col_int ,
1115   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1116   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1117   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1118 }

1119 \keys_define:nn { NiceMatrix / pNiceArray }
1120 {
1121   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1122   last-col .code:n = \tl_if_empty:nF { #1 }
1123                       { \@@_error:n { last-col-non-empty-for-NiceArray } }
1124                       \int_zero:N \l_@@_last_col_int ,
1125   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1126   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1127   delimiters / color .value_required:n = true ,
1128   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1129   delimiters / max-width .default:n = true ,
1130   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1131   delimiters .value_required:n = true ,
1132   small .bool_set:N = \l_@@_small_bool ,
1133   small .value_forbidden:n = true ,
```

```

1134   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1135   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1136   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1137 }

```

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

```

1138 \keys_define:nn { NiceMatrix / NiceTabular }
1139 {

```

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.

```

1140   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1141   \bool_set_true:N \l_@@_width_used_bool ,
1142   width .value_required:n = true ,
1143   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1144   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1145   tabularnote .value_required:n = true ,
1146   caption .tl_set:N = \l_@@_caption_tl ,
1147   caption .value_required:n = true ,
1148   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1149   short-caption .value_required:n = true ,
1150   label .tl_set:N = \l_@@_label_tl ,
1151   label .value_required:n = true ,
1152   last-col .code:n = \tl_if_empty:nF {#1}
1153   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1154   \int_zero:N \l_@@_last_col_int ,
1155   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1156   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1157   unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1158 }

```

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
```

```

1159 \keys_define:nn { NiceMatrix / CodeAfter }
1160 {
1161   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1162   delimiters / color .value_required:n = true ,
1163   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1164   rules .value_required:n = true ,
1165   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1166   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1167   sub-matrix .value_required:n = true ,
1168   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1169 }

```

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}).

```

1170 \cs_new_protected:Npn \@@_cell_begin:w
1171 {

```

`\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.

```
1172 \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).

```
1173 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1174 \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.

```
1175 \int_compare:nNnT \c@jCol = \c_one_int
1176 { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }
```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` is in the `\@@_cell_end:`.

```
1177 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1178 \@@_tuning_not_tabular_begin:
1179 \@@_tuning_first_row:
1180 \@@_tuning_last_row:
1181 \g_@@_row_style_tl
1182 }
```

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

```
1183 \cs_new_protected:Npn \@@_tuning_first_row:
1184 {
1185   \int_if_zero:nT \c@iRow
1186   {
1187     \int_compare:nNnT \c@jCol > \c_zero_int
1188     {
1189       \l_@@_code_for_first_row_tl
1190       \xglobal \colorlet { nicematrix-first-row } { . }
1191     }
1192   }
1193 }
```

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

```
1194 \cs_new_protected:Npn \@@_tuning_last_row:
1195 {
1196   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1197   {
1198     \l_@@_code_for_last_row_tl
1199     \xglobal \colorlet { nicematrix-last-row } { . }
1200   }
1201 }
```

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

```
1202 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1203 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1204 {
1205   \c_math_toggle_token
```

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

```
1206 \@@_tuning_key_small:
1207 }
1208 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token
```

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

```

1209 \cs_new_protected:Npn \@@_begin_of_row:
1210 {
1211   \int_gincr:N \c@iRow
1212   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1213   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1214   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1215   \pgfpicture
1216   \pgfrememberpicturepositiononpagetrue
1217   \pgfcoordinate
1218   { \@@_env: - row - \int_use:N \c@iRow - base }
1219   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1220   \str_if_empty:NF \l_@@_name_str
1221   {
1222     \pgfnodealias
1223     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1224     { \@@_env: - row - \int_use:N \c@iRow - base }
1225   }
1226   \endpgfpicture
1227 }

```

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.

```

1228 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1229 {
1230   \int_if_zero:nTF \c@iRow
1231   {
1232     \dim_gset:Nn \g_@@_dp_row_zero_dim
1233     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1234     \dim_gset:Nn \g_@@_ht_row_zero_dim
1235     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1236   }
1237   {
1238     \int_compare:nNnT \c@iRow = \c_one_int
1239     {
1240       \dim_gset:Nn \g_@@_ht_row_one_dim
1241       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1242     }
1243   }
1244 }

1245 \cs_new_protected:Npn \@@_rotate_cell_box:
1246 {
1247   \box_rotate:Nn \l_@@_cell_box { 90 }
1248   \bool_if:NTF \g_@@_rotate_c_bool
1249   {
1250     \hbox_set:Nn \l_@@_cell_box
1251     {
1252       \c_math_toggle_token
1253       \vcenter { \box_use:N \l_@@_cell_box }
1254       \c_math_toggle_token
1255     }
1256   }
1257   {
1258     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1259     {
1260       \vbox_set_top:Nn \l_@@_cell_box
1261       {

```

```

1262         \vbox_to_zero:n { }
1263         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1264         \box_use:N \l_@@_cell_box
1265     }
1266 }
1267 }
1268 \bool_gset_false:N \g_@@_rotate_bool
1269 \bool_gset_false:N \g_@@_rotate_c_bool
1270 }
1271 \cs_new_protected:Npn \@@_adjust_size_box:
1272 {
1273     \dim_compare:nNt \g_@@_blocks_wd_dim > \c_zero_dim
1274     {
1275         \box_set_wd:Nn \l_@@_cell_box
1276         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1277         \dim_gzero:N \g_@@_blocks_wd_dim
1278     }
1279     \dim_compare:nNt \g_@@_blocks_dp_dim > \c_zero_dim
1280     {
1281         \box_set_dp:Nn \l_@@_cell_box
1282         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1283         \dim_gzero:N \g_@@_blocks_dp_dim
1284     }
1285     \dim_compare:nNt \g_@@_blocks_ht_dim > \c_zero_dim
1286     {
1287         \box_set_ht:Nn \l_@@_cell_box
1288         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1289         \dim_gzero:N \g_@@_blocks_ht_dim
1290     }
1291 }
1292 \cs_new_protected:Npn \@@_cell_end:
1293 {

```

The following command is nullified in the tabulars.

```

1294     \@@_tuning_not_tabular_end:
1295     \hbox_set_end:
1296     \@@_cell_end_i:
1297 }
1298 \cs_new_protected:Npn \@@_cell_end_i:
1299 {

```

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.

```

1300     \g_@@_cell_after_hook_tl
1301     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1302     \@@_adjust_size_box:
1303     \box_set_ht:Nn \l_@@_cell_box
1304     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1305     \box_set_dp:Nn \l_@@_cell_box
1306     { \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”).

```

1307     \@@_update_max_cell_width:

```

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

```

1308     \@@_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.

```

1309   \bool_if:NTF \g_@@_empty_cell_bool
1310   { \box_use_drop:N \l_@@_cell_box }
1311   {
1312     \bool_if:NTF \g_@@_not_empty_cell_bool
1313     \@@_node_for_cell:
1314     {
1315       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1316       \@@_node_for_cell:
1317       { \box_use_drop:N \l_@@_cell_box }
1318     }
1319   }
1320   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1321   \bool_gset_false:N \g_@@_empty_cell_bool
1322   \bool_gset_false:N \g_@@_not_empty_cell_bool
1323 }

```

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

```

1324 \cs_new_protected:Npn \@@_update_max_cell_width:
1325 {
1326   \dim_gset:Nn \g_@@_max_cell_width_dim
1327   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1328 }

```

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`).

```

1329 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1330 {
1331   \@@_math_toggle:
1332   \hbox_set_end:
1333   \bool_if:NF \g_@@_rotate_bool
1334   {
1335     \hbox_set:Nn \l_@@_cell_box
1336     {
1337       \makebox [ \l_@@_col_width_dim ] [ s ]
1338       { \hbox_unpack_drop:N \l_@@_cell_box }
1339     }
1340   }
1341   \@@_cell_end_i:
1342 }

```

```

1343 \pgfset
1344 {
1345   nicematrix / cell-node /.style =
1346   {
1347     inner-sep = \c_zero_dim ,
1348     minimum-width = \c_zero_dim

```

```

1349   }
1350 }

```

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

```

1351 \cs_new_protected:Npn \@@_node_for_cell:
1352 {
1353   \pgfpicture
1354   \pgfsetbaseline \c_zero_dim
1355   \pgfrememberpicturerepositiononpagetrue
1356   \pgfset { nicematrix / cell-node }
1357   \pgfnode
1358   { rectangle }
1359   { base }
1360   {

```

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

```

1361     \set@color
1362     \box_use_drop:N \l_@@_cell_box
1363   }
1364   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1365   { \l_@@_pgf_node_code_tl }
1366   \str_if_empty:NF \l_@@_name_str
1367   {
1368     \pgfnodealias
1369     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1370     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1371   }
1372   \endpgfpicture
1373 }

```

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.

```

1374 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1375 {
1376   \cs_new_protected:Npn \@@_patch_node_for_cell:
1377   {
1378     \hbox_set:Nn \l_@@_cell_box
1379     {
1380       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1381       \hbox_overlap_left:n
1382       {
1383         \pgfsys@markposition
1384         { \@@_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.

```

1385         #1
1386       }
1387       \box_use:N \l_@@_cell_box
1388       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1389       \hbox_overlap_left:n
1390       {
1391         \pgfsys@markposition
1392         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1393       }
1394     }
1395   }
1396 }
1397 }

```


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

```

1398 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1399 {
1400   \@@_patch_node_for_cell:n
1401     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1402 }
1403 { \@@_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).

```

1404 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1405 {
1406   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1407     { g_@@_ #2 _ lines _ tl }
1408   {
1409     \use:c { @@ _ draw _ #2 : nnn }
1410     { \int_use:N \c@iRow }
1411     { \int_use:N \c@jCol }
1412     { \exp_not:n { #3 } }
1413   }
1414 }

1415 \cs_new_protected:Npn \@@_array:
1416 {
1417   % \begin{macrocode}
1418   \dim_set:Nn \col@sep
1419     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1420   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1421     { \cs_set_nopar:Npn \@halignto { } }
1422     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1423 \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.

```

1424 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1425 }

```

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. However, since version 2.6a (version for the Tagging Project), `array` uses `\ar@ialign` instead of `\ialign`. In that case, of course, you do a saving of `\ar@ialign`.

```

1426 \bool_if:NTF \c_@@_tagging_array_bool
1427 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1428 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1429 \cs_new_protected:Npn \@@_create_row_node:
1430 {
1431   \int_compare:nNtT \c@iRow > \g_@@_last_row_node_int
1432   {
1433     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1434     \@@_create_row_node_i:
1435   }
1436 }
1437 \cs_new_protected:Npn \@@_create_row_node_i:
1438 {

```

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

```

1439   \hbox
1440   {
1441     \bool_if:NT \l_@@_code_before_bool
1442     {
1443       \vtop
1444       {
1445         \skip_vertical:N 0.5\arrayrulewidth
1446         \pgfsys@markposition
1447         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1448         \skip_vertical:N -0.5\arrayrulewidth
1449       }
1450     }
1451     \pgfpicture
1452     \pgfrememberpicturepositiononpagetrue
1453     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1454     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1455     \str_if_empty:NF \l_@@_name_str
1456     {
1457       \pgfnodealias
1458       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1459       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1460     }
1461     \endpgfpicture
1462   }
1463 }

```

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

```

1464 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1465 \cs_new_protected:Npn \@@_everycr_i:
1466 {
1467   \bool_if:NT \c_@@_tagging_array_bool
1468   {
1469     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1470     \tbl_update_cell_data_for_next_row:
1471   }
1472   \int_gzero:N \c@jCol
1473   \bool_gset_false:N \g_@@_after_col_zero_bool
1474   \bool_if:NF \g_@@_row_of_col_done_bool
1475   {
1476     \@@_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).

```

1477   \tbl_if_empty:NF \l_@@_hlines_clist

```

```

1478     {
1479       \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1480       {
1481         \exp_args:NNe
1482         \clist_if_in:NnT
1483         \l_@@_hlines_clist
1484         { \int_eval:n { \c@iRow + 1 } }
1485       }
1486     {

```

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.

```

1487       \int_compare:nNnT \c@iRow > { -1 }
1488       {
1489         \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@`.

```

1490         { \hrule height \arrayrulewidth width \c_zero_dim }
1491       }
1492     }
1493   }
1494 }
1495 }

```

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

```

1496 \cs_set_protected:Npn \@@_renew_dots:
1497 {
1498   \cs_set_eq:NN \ldots \@@_Ldots
1499   \cs_set_eq:NN \cdots \@@_Cdots
1500   \cs_set_eq:NN \vdots \@@_Vdots
1501   \cs_set_eq:NN \ddots \@@_Ddots
1502   \cs_set_eq:NN \iddots \@@_Iddots
1503   \cs_set_eq:NN \dots \@@_Ldots
1504   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1505 }
1506 \cs_new_protected:Npn \@@_test_color_inside:
1507 {
1508   \bool_if:NF \l_@@_color_inside_bool
1509   {

```

We will issue an error only during the first run.

```

1510     \bool_if:NF \g_@@_aux_found_bool
1511     { \@@_error:n { without~color~inside } }
1512   }
1513 }
1514 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1515 \hook_gput_code:nnn { begindocument } { . }
1516 {
1517   \IfPackageLoadedTF { colortbl }
1518   {
1519     \cs_set_protected:Npn \@@_redefine_everycr:
1520     {
1521       \CT@everycr
1522       {
1523         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1524         \@@_everycr:
1525       }
1526     }
1527   }
1528 }
1529 }

```

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's 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 redefinition⁴.

```

1530 \hook_gput_code:nnn { begindocument } { . }
1531 {
1532   \IfPackageLoadedTF { booktabs }
1533   {
1534     \cs_new_protected:Npn \@_patch_booktabs:
1535       { \tl_put_left:Nn \@BTnormal \@_create_row_node_i: }
1536   }
1537   { \cs_new_protected:Npn \@_patch_booktabs: { } }
1538 }

```

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`.

```

1539 \cs_new_protected:Npn \@_some_initialization:
1540 {
1541   \dim_gzero_new:N \g_@@_dp_row_zero_dim
1542   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1543   \dim_gzero_new:N \g_@@_ht_row_zero_dim
1544   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1545   \dim_gzero_new:N \g_@@_ht_row_one_dim
1546   \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1547   \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1548   \dim_gzero_new:N \g_@@_ht_last_row_dim
1549   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1550   \dim_gzero_new:N \g_@@_dp_last_row_dim
1551   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1552 }

```

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

```

1553 \cs_new_protected:Npn \@_pre_array_ii:
1554 {

```

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

```

1555   \int_gzero:N \g_@@_total_X_weight_int
1556   \@@_expand_clist:N \l_@@_hlines_clist
1557   \@@_expand_clist:N \l_@@_vlines_clist
1558   \@_patch_booktabs:
1559   \box_clear_new:N \l_@@_cell_box
1560   \normalbaselines

```

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

```

1561   \bool_if:NT \l_@@_small_bool
1562   {

```

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

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

```

1563     \cs_set_nopar:Npn \arraystretch { 0.47 }
1564     \dim_set:Nn \arraycolsep { 1.45 pt }

By default, \@@_small_scripstyle: is null.

1565     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1566 }

1567 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1568 {
1569     \tl_put_right:Nn \@@_begin_of_row:
1570     {
1571         \pgfsys@markposition
1572         { \@@_env: - row - \int_use:N \c@iRow - base }
1573     }
1574 }

```

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`.

```

1575     \bool_if:NTF \c_@@_tagging_array_bool
1576     {
1577         \cs_set_nopar:Npn \ar@ialign
1578         {
1579             \tbl_init_cell_data_for_table:
1580             \@@_redefine_everycr:
1581             \tabskip = \c_zero_skip
1582             \@@_some_initialization:

```

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

```

1583         \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1584         \halign
1585     }
1586 }

```

The following part will be deleted when we will delete the boolean `\c_@@_tagging_array_bool` (when we consider the version 2.6a of `array` is required).

```

1587 {
1588     \cs_set_nopar:Npn \ialign
1589     {
1590         \@@_redefine_everycr:
1591         \tabskip = \c_zero_skip
1592         \@@_some_initialization:
1593         \cs_set_eq:NN \ialign \@@_old_ialign:
1594         \halign
1595     }
1596 }

```

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).

```

1597     \cs_set_eq:NN \@@_old_ldots \ldots
1598     \cs_set_eq:NN \@@_old_cdots \cdots
1599     \cs_set_eq:NN \@@_old_vdots \vdots
1600     \cs_set_eq:NN \@@_old_ddots \ddots
1601     \cs_set_eq:NN \@@_old_iddots \iddots
1602     \bool_if:NTF \l_@@_standard_cline_bool
1603     { \cs_set_eq:NN \cline \@@_standard_cline }
1604     { \cs_set_eq:NN \cline \@@_cline }
1605     \cs_set_eq:NN \Ldots \@@_Ldots
1606     \cs_set_eq:NN \Cdots \@@_Cdots
1607     \cs_set_eq:NN \Vdots \@@_Vdots
1608     \cs_set_eq:NN \Ddots \@@_Ddots

```

```

1609 \cs_set_eq:NN \Iddots \@@_Iddots
1610 \cs_set_eq:NN \Hline \@@_Hline:
1611 \cs_set_eq:NN \Hspace \@@_Hspace:
1612 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1613 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1614 \cs_set_eq:NN \Block \@@_Block:
1615 \cs_set_eq:NN \rotate \@@_rotate:
1616 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1617 \cs_set_eq:NN \dotfill \@@_dotfill:
1618 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1619 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1620 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1621 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1622 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1623   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1624 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1625 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1626 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1627 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1628 \int_compare:nNt \l_@@_first_row_int > \c_zero_int
1629   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1630 \int_compare:nNt \l_@@_last_row_int < \c_zero_int
1631   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1632 \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. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```

1633 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1634 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1635   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1636 \@@_revert_colortbl:

```

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

```

1637 \tl_if_exist:NT \l_@@_note_in_caption_tl
1638   {
1639     \tl_if_empty:NF \l_@@_note_in_caption_tl
1640     {
1641       \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1642       \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1643     }
1644   }

```

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

```

1645 \seq_gclear:N \g_@@_multicolumn_cells_seq
1646 \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).

```

1647 \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`).

```

1648 \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.

```

1649 \int_gzero_new:N \g_@@_col_total_int
1650 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1651 \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.

```

1652 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1653 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1654 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1655 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1656 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1657 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1658 \tl_gclear:N \g_nicematrix_code_before_tl
1659 \tl_gclear:N \g_@@_pre_code_before_tl
1660 }

```

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

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

```

1661 \cs_new_protected:Npn \@@_pre_array:
1662 {
1663   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1664   \int_gzero_new:N \c@iRow
1665   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1666   \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).

```

1667 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1668 {
1669   \bool_set_true:N \l_@@_last_row_without_value_bool
1670   \bool_if:NT \g_@@_aux_found_bool
1671     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1672 }
1673 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1674 {
1675   \bool_if:NT \g_@@_aux_found_bool
1676     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1677 }

```

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”.

```

1678 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1679 {
1680   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1681   {
1682     \dim_gset:Nn \g_@@_ht_last_row_dim
1683     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1684     \dim_gset:Nn \g_@@_dp_last_row_dim
1685     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1686   }
1687 }

```

```

1688 \seq_gclear:N \g_@@_cols_vlism_seq
1689 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1690 \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.

```

1691 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1692 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1693 \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.

```

1694 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1695 \@@_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.

```

1696 \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]`).

```

1697 \dim_zero_new:N \l_@@_left_delim_dim
1698 \dim_zero_new:N \l_@@_right_delim_dim
1699 \bool_if:NTF \g_@@_delims_bool
1700 {

```

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

```

1701 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1702 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1703 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1704 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1705 }
1706 {
1707 \dim_gset:Nn \l_@@_left_delim_dim
1708 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1709 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1710 }

```

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).

```

1711 \hbox_set:Nw \l_@@_the_array_box
1712 \skip_horizontal:N \l_@@_left_margin_dim
1713 \skip_horizontal:N \l_@@_extra_left_margin_dim
1714 \c_math_toggle_token
1715 \bool_if:NTF \l_@@_light_syntax_bool
1716 { \use:c { @@-light-syntax } }
1717 { \use:c { @@-normal-syntax } }
1718 }

```


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

```

1719 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1720 {
1721   \tl_set:Nn \l_tmpa_tl { #1 }
1722   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1723     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1724   \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1725   \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...

```

1726   \@@_pre_array:
1727 }

```

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).

```

1728 \cs_new_protected:Npn \@@_pre_code_before:
1729 {

```

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).

```

1730   \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1731   \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1732   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1733   \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.

```

1734   \pgfsys@markposition { \@@_env: - position }
1735   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1736   \pgfpicture
1737   \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1738   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1739   {
1740     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1741     \pgfcoordinate { \@@_env: - row - ##1 }
1742     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1743   }

```

Now, the recreation of the `col` nodes.

```

1744   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1745   {
1746     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1747     \pgfcoordinate { \@@_env: - col - ##1 }
1748     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1749   }

```

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

```

1750   \@@_create_diag_nodes:

```

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

```
1751 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1752 \endpgfpicture
```

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

```
1753 \@@_create_blocks_nodes:
1754 \IfPackageLoadedTF { tikz }
1755 {
1756   \tikzset
1757   {
1758     every-picture / .style =
1759     { overlay , name-prefix = \@@_env: - }
1760   }
1761 }
1762 { }
1763 \cs_set_eq:NN \cellcolor \@@_cellcolor
1764 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1765 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1766 \cs_set_eq:NN \rowcolor \@@_rowcolor
1767 \cs_set_eq:NN \rowcolors \@@_rowcolors
1768 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1769 \cs_set_eq:NN \arraycolor \@@_arraycolor
1770 \cs_set_eq:NN \columncolor \@@_columncolor
1771 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1772 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1773 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1774 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1775 }

1776 \cs_new_protected:Npn \@@_exec_code_before:
1777 {
1778   \seq_gclear_new:N \g_@@_colors_seq
```

The sequence `\g_@@_colors_seq` will always contain as first element the special color `nocolor`: when that color is used, no color will be applied in the corresponding cells by the other coloring commands of `nicematrix`.

```
1779 \@@_add_to_colors_seq:nn { { nocolor } } { }
1780 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1781 \group_begin:
```

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

```
1782 \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`).

```
1783 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1784 { \@@_rescan_for_spanish:N \l_@@_code_before_tl }
```

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

```
1785 \exp_last_unbraced:NV \@@_CodeBefore_keys:
1786 \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.

```

1787     \@@_actually_color:
1788     \l_@@_code_before_tl
1789     \q_stop
1790     \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1791     \group_end:
1792     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1793     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1794 }

1795 \keys_define:nn { NiceMatrix / CodeBefore }
1796 {
1797     create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1798     create-cell-nodes .default:n = true ,
1799     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1800     sub-matrix .value_required:n = true ,
1801     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1802     delimiters / color .value_required:n = true ,
1803     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1804 }

1805 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1806 {
1807     \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1808     \@@_CodeBefore:w
1809 }

```

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.

```

1810 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1811 {
1812     \bool_if:NT \g_@@_aux_found_bool
1813     {
1814         \@@_pre_code_before:
1815         #1
1816     }
1817 }

```

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.

```

1818 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1819 {
1820     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1821     {
1822         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1823         \pgfcoordinate { \@@_env: - row - ##1 - base }
1824         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1825         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1826         {
1827             \cs_if_exist:cT
1828             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1829             {
1830                 \pgfsys@getposition
1831                 { \@@_env: - ##1 - #####1 - NW }
1832                 \@@_node_position:
1833                 \pgfsys@getposition
1834                 { \@@_env: - ##1 - #####1 - SE }

```

```

1835         \@@_node_position_i:
1836         \@@_pgf_rect_node:nnn
1837         { \@@_env: - ##1 - #####1 }
1838         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1839         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1840     }
1841 }
1842 }
1843 \int_step_inline:nn \c@iRow
1844 {
1845     \pgfnodealias
1846     { \@@_env: - ##1 - last }
1847     { \@@_env: - ##1 - \int_use:N \c@jCol }
1848 }
1849 \int_step_inline:nn \c@jCol
1850 {
1851     \pgfnodealias
1852     { \@@_env: - last - ##1 }
1853     { \@@_env: - \int_use:N \c@iRow - ##1 }
1854 }
1855 \@@_create_extra_nodes:
1856 }

1857 \cs_new_protected:Npn \@@_create_blocks_nodes:
1858 {
1859     \pgfpicture
1860     \pgf@relevantforpicturesizefalse
1861     \pgfrememberpicturepositiononpagetrue
1862     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1863     { \@@_create_one_block_node:nnnnn ##1 }
1864     \endpgfpicture
1865 }

```

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

```

1866 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1867 {
1868     \tl_if_empty:nF { #5 }
1869     {
1870         \@@_qpoint:n { col - #2 }
1871         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1872         \@@_qpoint:n { #1 }
1873         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1874         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1875         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1876         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1877         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1878         \@@_pgf_rect_node:nnnnn
1879         { \@@_env: - #5 }
1880         { \dim_use:N \l_tmpa_dim }
1881         { \dim_use:N \l_tmpb_dim }
1882         { \dim_use:N \l_@@_tmpc_dim }
1883         { \dim_use:N \l_@@_tmpd_dim }
1884     }
1885 }

1886 \cs_new_protected:Npn \@@_patch_for_revtext:
1887 {

```

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).

```

1888 \cs_set_eq:NN \@addamp \@addamp@LaTeX
1889 \cs_set_eq:NN \insert@column \insert@column@array
1890 \cs_set_eq:NN \@classx \@classx@array
1891 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1892 \cs_set_eq:NN \@arraycr \@arraycr@array
1893 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1894 \cs_set_eq:NN \array \array@array
1895 \cs_set_eq:NN \@array \@array@array
1896 \cs_set_eq:NN \@tabular \@tabular@array
1897 \cs_set_eq:NN \@mkpream \@mkpream@array
1898 \cs_set_eq:NN \endarray \endarray@array
1899 \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1900 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1901 }

```

11 The environment `{NiceArrayWithDelims}`

```

1902 \NewDocumentEnvironment { NiceArrayWithDelims }
1903 { m m O { } m ! O { } t \CodeBefore }
1904 {
1905 \bool_if:NT \c_@@_revtex_bool \@_patch_for_revtex:
1906 \@_provide_pgfsyspdfmark:
1907 \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.

```

1908 \bgroup

1909 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1910 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1911 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1912 \int_gzero:N \g_@@_block_box_int
1913 \dim_zero:N \g_@@_width_last_col_dim
1914 \dim_zero:N \g_@@_width_first_col_dim
1915 \bool_gset_false:N \g_@@_row_of_col_done_bool
1916 \str_if_empty:NT \g_@@_name_env_str
1917 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1918 \bool_if:NTF \l_@@_tabular_bool
1919 \mode_leave_vertical:
1920 \@_test_if_math_mode:
1921 \bool_if:NT \l_@@_in_env_bool { \@_fatal:n { Yet~in~env } }
1922 \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.

```

1923 \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.

```

1924 \cs_if_exist:NT \tikz@library@external@loaded
1925 {

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1926      \tikzexternaldisable
1927      \cs_if_exist:NT \ifstandalone
1928      { \tikzset { external / optimize = false } }
1929  }

```

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

```

1930      \int_gincr:N \g_@@_env_int
1931      \bool_if:NF \l_@@_block_auto_columns_width_bool
1932      { \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`).

```

1933      \seq_gclear:N \g_@@_blocks_seq
1934      \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1935      \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1936      \seq_gclear:N \g_@@_pos_of_xdots_seq
1937      \tl_gclear_new:N \g_@@_code_before_tl
1938      \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.

```

1939      \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1940      {
1941          \bool_gset_true:N \g_@@_aux_found_bool
1942          \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1943      }
1944      { \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.

```

1945      \tl_gclear:N \g_@@_aux_tl
1946      \tl_if_empty:NF \g_@@_code_before_tl
1947      {
1948          \bool_set_true:N \l_@@_code_before_bool
1949          \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1950      }
1951      \tl_if_empty:NF \g_@@_pre_code_before_tl
1952      { \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`.

```

1953      \bool_if:NTF \g_@@_delims_bool
1954      { \keys_set:nn { NiceMatrix / pNiceArray } }
1955      { \keys_set:nn { NiceMatrix / NiceArray } }
1956      { #3 , #5 }

1957      \@@_set_CT@arc@:o \l_@@_rules_color_tl

```

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

```

1958      \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1959  }

```

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

```

1960      {
1961          \bool_if:NTF \l_@@_light_syntax_bool

```

```

1962     { \use:c { end @@-light-syntax } }
1963     { \use:c { end @@-normal-syntax } }
1964     \c_math_toggle_token
1965     \skip_horizontal:N \l_@@_right_margin_dim
1966     \skip_horizontal:N \l_@@_extra_right_margin_dim
1967     \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.

```

1968     \bool_if:NT \l_@@_width_used_bool
1969     {
1970         \int_if_zero:nT \g_@@_total_X_weight_int
1971         { \@@_error_or_warning:n { width-without-X-columns } }
1972     }

```

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 .

```

1973     \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1974     {
1975         \tl_gput_right:Nx \g_@@_aux_tl
1976         {
1977             \bool_set_true:N \l_@@_X_columns_aux_bool
1978             \dim_set:Nn \l_@@_X_columns_dim
1979             {
1980                 \dim_compare:nNnTF
1981                 {
1982                     \dim_abs:n
1983                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1984                 }
1985                 <
1986                 { 0.001 pt }
1987                 { \dim_use:N \l_@@_X_columns_dim }
1988                 {
1989                     \dim_eval:n
1990                     {
1991                         ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1992                         / \int_use:N \g_@@_total_X_weight_int
1993                         + \l_@@_X_columns_dim
1994                     }
1995                 }
1996             }
1997         }
1998     }

```

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

```

1999     \int_compare:nNnT \l_@@_last_row_int > { -2 }
2000     {
2001         \bool_if:NF \l_@@_last_row_without_value_bool
2002         {
2003             \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2004             {
2005                 \@@_error:n { Wrong-last-row }
2006                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2007             }
2008         }
2009     }

```

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

```

2010 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2011 \bool_if:NTF \g_@@_last_col_found_bool
2012 { \int_gdecr:N \c@jCol }
2013 {
2014 \int_compare:nNnT \l_@@_last_col_int > { -1 }
2015 { \@@_error:n { last~col~not~used } }
2016 }

```

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

```

2017 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2018 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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

```

2019 \int_if_zero:nT \l_@@_first_col_int
2020 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2021 \bool_if:nTF { ! \g_@@_delims_bool }
2022 {
2023 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2024 \@@_use_arraybox_with_notes_c:
2025 {
2026 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2027 \@@_use_arraybox_with_notes_b:
2028 \@@_use_arraybox_with_notes:
2029 }
2030 }

```

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).

```

2031 {
2032 \int_if_zero:nTF \l_@@_first_row_int
2033 {
2034 \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2035 \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2036 }
2037 { \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

```

2038 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2039 {
2040 \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2041 \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2042 }
2043 { \dim_zero:N \l_tmpb_dim }
2044 \hbox_set:Nn \l_tmpa_box
2045 {
2046 \c_math_toggle_token
2047 \@@_color:o \l_@@_delimiters_color_tl
2048 \exp_after:wN \left \g_@@_left_delim_tl
2049 \vcenter
2050 {

```

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.

```

2051 \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2052 \hbox

```

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

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


```

2053     {
2054         \bool_if:NTF \l_@@_tabular_bool
2055         { \skip_horizontal:N -\tabcolsep }
2056         { \skip_horizontal:N -\arraycolsep }
2057         \@@_use_arraybox_with_notes_c:
2058         \bool_if:NTF \l_@@_tabular_bool
2059         { \skip_horizontal:N -\tabcolsep }
2060         { \skip_horizontal:N -\arraycolsep }
2061     }

```

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

```

2062         \skip_vertical:N -\l_tmpb_dim
2063         \skip_vertical:N \arrayrulewidth
2064     }
2065     \exp_after:wN \right \g_@@_right_delim_tl
2066     \c_math_toggle_token
2067 }

```

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.

```

2068     \bool_if:NTF \l_@@_delimiters_max_width_bool
2069     {
2070         \@@_put_box_in_flow_bis:nn
2071         \g_@@_left_delim_tl
2072         \g_@@_right_delim_tl
2073     }
2074     \@@_put_box_in_flow:
2075 }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 89).

```

2076     \bool_if:NT \g_@@_last_col_found_bool
2077     { \skip_horizontal:N \g_@@_width_last_col_dim }
2078     \bool_if:NT \l_@@_preamble_bool
2079     {
2080         \int_compare:nNtT \c@jCol < \g_@@_static_num_of_col_int
2081         { \@@_warning_gredirect_none:n { columns-not-used } }
2082     }
2083     \@@_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.

```

2084     \egroup

```

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

```

2085     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2086     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2087     \iow_now:Nx \@mainaux
2088     {
2089         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2090         { \exp_not:o \g_@@_aux_tl }
2091     }
2092     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2093     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2094 }

```

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.

```

2095 \cs_new_protected:Npn \@@_transform_preamble:
2096 {
2097   \@@_transform_preamble_i:
2098   \@@_transform_preamble_ii:
2099 }
2100 \cs_new_protected:Npn \@@_transform_preamble_i:
2101 {
2102   \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`).

```

2103   \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.

```

2104   \bool_gset_false:N \g_tmpb_bool

```

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

```

2105   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2106   \int_zero:N \l_tmpa_int
2107   \tl_gclear:N \g_@@_array_preamble_tl
2108   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2109   {
2110     \tl_gset:Nn \g_@@_array_preamble_tl
2111     { ! { \skip_horizontal:N \arrayrulewidth } }
2112   }
2113   {
2114     \clist_if_in:NnT \l_@@_vlines_clist 1
2115     {
2116       \tl_gset:Nn \g_@@_array_preamble_tl
2117       { ! { \skip_horizontal:N \arrayrulewidth } }
2118     }
2119   }

```

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

```

2120   \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2121   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2122   \@@_replace_columncolor:
2123 }

2124 \hook_gput_code:nnn { begindocument } { . }
2125 {
2126   \IfPackageLoadedTF { colortbl }
2127   {
2128     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2129     \cs_new_protected:Npn \@@_replace_columncolor:
2130     {
2131       \regex_replace_all:NnN
2132       \c_@@_columncolor_regex
2133       { \c { @@_columncolor_preamble } }

```

```

2134         \g_@@_array_preamble_tl
2135     }
2136 }
2137 {
2138     \cs_new_protected:Npn \@@_replace_columncolor:
2139     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2140 }
2141 }

```

```

2142 \cs_new_protected:Npn \@@_transform_preamble_ii:
2143 {

```

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

```

2144     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2145     {
2146         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2147         { \bool_gset_true:N \g_@@_delims_bool }
2148     }
2149     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2150     \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).

```

2151     \int_if_zero:nTF \l_@@_first_col_int
2152     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2153     {
2154         \bool_if:NF \g_@@_delims_bool
2155         {
2156             \bool_if:NF \l_@@_tabular_bool
2157             {
2158                 \tl_if_empty:NT \l_@@_vlines_clist
2159                 {
2160                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2161                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2162                 }
2163             }
2164         }
2165     }
2166     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2167     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2168     {
2169         \bool_if:NF \g_@@_delims_bool
2170         {
2171             \bool_if:NF \l_@@_tabular_bool
2172             {
2173                 \tl_if_empty:NT \l_@@_vlines_clist
2174                 {
2175                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2176                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2177                 }
2178             }
2179         }
2180     }

```

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`).

```

2181     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2182     {
2183         \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2184         { > { \@@_error_too_much_cols: } 1 }
2185     }
2186 }

```

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.

```

2187 \cs_new_protected:Npn \@@_rec_preamble:n #1
2188 {

```

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

```

2189     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2190     { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2191     {

```

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

```

2192     \cs_if_exist:cTF { NC @ find @ #1 }
2193     {
2194         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2195         \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2196     }
2197     {
2198         \tl_if_eq:nnT { #1 } { S }
2199         { \@@_fatal:n { unknown~column~type~S } }
2200         { \@@_fatal:nn { unknown~column~type } { #1 } }
2201     }
2202 }
2203 }

```

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

```

2204 \cs_new:Npn \@@_c #1
2205 {
2206     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2207     \tl_gclear:N \g_@@_pre_cell_tl
2208     \tl_gput_right:Nn \g_@@_array_preamble_tl
2209     { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2210     \int_gincr:N \c@jCol
2211     \@@_rec_preamble_after_col:n
2212 }

2213 \cs_new:Npn \@@_l #1
2214 {
2215     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2216     \tl_gclear:N \g_@@_pre_cell_tl
2217     \tl_gput_right:Nn \g_@@_array_preamble_tl
2218     {
2219         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2220         l
2221         < \@@_cell_end:
2222     }
2223     \int_gincr:N \c@jCol
2224     \@@_rec_preamble_after_col:n
2225 }

```

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

```

2226 \cs_new:Npn \@@_r #1
2227 {
2228   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2229   \tl_gclear:N \g_@@_pre_cell_tl
2230   \tl_gput_right:Nn \g_@@_array_preamble_tl
2231   {
2232     > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2233     r
2234     < \@@_cell_end:
2235   }
2236   \int_gincr:N \c@jCol
2237   \@@_rec_preamble_after_col:n
2238 }

```

For ! and @

```

2239 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2240 {
2241   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2242   \@@_rec_preamble:n
2243 }
2244 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2245 \cs_new:cpn { @@ _ | } #1
2246 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2247   \int_incr:N \l_tmpa_int
2248   \@@_make_preamble_i_i:n
2249 }
2250 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2251 {
2252   \str_if_eq:nnTF { #1 } |
2253   { \use:c { @@ _ | } | }
2254   { \@@_make_preamble_i_ii:nn { } #1 }
2255 }
2256 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2257 {
2258   \str_if_eq:nnTF { #2 } [
2259   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2260   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2261 }
2262 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2263 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2264 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2265 {
2266   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2267   \tl_gput_right:Nx \g_@@_array_preamble_tl
2268   {

```

Here, the command \dim_eval:n is mandatory.

```

2269   \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2270 }
2271 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2272 {
2273   \@@_vline:n
2274   {
2275     position = \int_eval:n { \c@jCol + 1 } ,
2276     multiplicity = \int_use:N \l_tmpa_int ,
2277     total-width = \dim_use:N \l_@@_rule_width_dim ,
2278     #2
2279   }

```

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.

```

2280     }
2281     \int_zero:N \l_tmpa_int
2282     \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2283     \@@_rec_preamble:n #1
2284   }

2285   \cs_new:cpn { @@ _ > } #1 #2
2286   {
2287     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2288     \@@_rec_preamble:n
2289   }

2290   \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.

```

2291 \keys_define:nn { nicematrix / p-column }
2292 {
2293   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2294   r .value_forbidden:n = true ,
2295   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2296   c .value_forbidden:n = true ,
2297   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2298   l .value_forbidden:n = true ,
2299   R .code:n =
2300     \IfPackageLoadedTF { ragged2e }
2301     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2302     {
2303       \@@_error_or_warning:n { ragged2e~not~loaded }
2304       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2305     } ,
2306   R .value_forbidden:n = true ,
2307   L .code:n =
2308     \IfPackageLoadedTF { ragged2e }
2309     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2310     {
2311       \@@_error_or_warning:n { ragged2e~not~loaded }
2312       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2313     } ,
2314   L .value_forbidden:n = true ,
2315   C .code:n =
2316     \IfPackageLoadedTF { ragged2e }
2317     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2318     {
2319       \@@_error_or_warning:n { ragged2e~not~loaded }
2320       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2321     } ,
2322   C .value_forbidden:n = true ,
2323   S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2324   S .value_forbidden:n = true ,
2325   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2326   p .value_forbidden:n = true ,
2327   t .meta:n = p ,
2328   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2329   m .value_forbidden:n = true ,
2330   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2331   b .value_forbidden:n = true ,
2332 }

```

For p, b and m.

```

2333 \cs_new:Npn \@@_p #1
2334 {
2335   \str_set:Nn \l_@@_vpos_col_str { #1 }
2336   \@@_make_preamble_ii_i:n
2337 }
2338 \cs_set_eq:NN \@@_b \@@_p
2339 \cs_set_eq:NN \@@_m \@@_p
2340 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2341 {
2342   \str_if_eq:nnTF { #1 } { [ ] }
2343   { \@@_make_preamble_ii_ii:w [ ] }
2344   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2345 }
2346 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2347 { \@@_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.

```

2348 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2349 {

```

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).

```

2350   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2351   \@@_keys_p_column:n { #1 }
2352   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2353 }
2354 \cs_new_protected:Npn \@@_keys_p_column:n #1
2355 { \keys_set:known { nicematrix / p-column } { #1 } \l_tmpa_tl }

```

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

```

2356 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2357 {
2358   \use:e
2359   {
2360     \@@_make_preamble_ii_v:nnnnnnnn
2361     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2362     { \dim_eval:n { #1 } }
2363     {

```

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

```

2364   \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2365   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2366   {
2367     \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2368     { \str_lowercase:V \l_@@_hpos_col_str }
2369   }
2370   \str_case:on \l_@@_hpos_col_str
2371   {
2372     c { \exp_not:N \centering }
2373     l { \exp_not:N \raggedright }
2374     r { \exp_not:N \raggedleft }
2375     C { \exp_not:N \Centering }
2376     L { \exp_not:N \RaggedRight }
2377     R { \exp_not:N \RaggedLeft }

```

```

2378     }
2379     #3
2380 }
2381 { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2382 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2383 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2384 { #2 }
2385 {
2386     \str_case:onF \l_@@_hpos_col_str
2387     {
2388         { j } { c }
2389         { si } { c }
2390     }

```

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

```

2391     { \str_lowercase:V \l_@@_hpos_col_str }
2392 }
2393 }

```

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

```

2394     \int_gincr:N \c_jCol
2395     \@@_rec_preamble_after_col:n
2396 }

```

#1 is the optional argument of `{minipage}` (or `{varwidth}`): `t` or `b`. Indeed, for the columns of type `m`, we use the value `b` here because there is a special post-action in order to center vertically the box (see #4).

#2 is the width of the `{minipage}` (or `{varwidth}`), that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing). It's also possible to put in that #3 some code to fix the value of `\l_@@_hpos_cell_tl` which will be available in each cell of the column.

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

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

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

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

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

```

2397 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2398 {
2399     \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2400     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2401     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2402     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2403     \tl_gclear:N \g_@@_pre_cell_tl
2404     \tl_gput_right:Nn \g_@@_array_preamble_tl
2405     {
2406         > {

```

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.

```

2407     \dim_set:Nn \l_@@_col_width_dim { #2 }
2408     \@@_cell_begin:w

```

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

```

2409     \use:c { #7 } [ #1 ] { #2 }

```

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

```

2410     \everypar
2411     {
2412         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2413         \everypar { }
2414     }

```


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

```
2415         #3
```

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

```
2416         \g_@@_row_style_tl
2417         \arraybackslash
2418         #5
2419     }
2420     #8
2421     < {
2422         #6
```

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

```
2423         \@finalstrut \@arstrutbox
2424         \use:c { end #7 }
```

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

```
2425         #4
2426         \@@_cell_end:
2427     }
2428 }
2429 }
```

```
2430 \str_new:N \c_@@_ignorespaces_str
2431 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2432 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }
```

In order to test whether a cell is empty, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstances, for example when `\collectcell` of `collcell` is used, the cell does not begin with `\ignorespaces`. In that case, we consider as not empty...

First, we test if the next token is `\ignorespaces` and it's not very easy...

```
2433 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2434 \cs_new_protected:Npn \@@_test_if_empty_i:
2435 {
2436     \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2437     \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2438     { \@@_test_if_empty:w }
2439 }
2440 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2441 {
2442     \peek_meaning:NT \unskip
2443     {
2444         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2445         {
2446             \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2447             \skip_horizontal:N \l_@@_col_width_dim
2448         }
2449     }
2450 }
2451 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2452 {
2453     \peek_meaning:NT \__siunitx_table_skip:n
2454     {
2455         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2456         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2457     }
2458 }
```

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 than the height of `\strutbox`, there is only one row.

```
2459 \cs_new_protected:Npn \@@_center_cell_box:
2460 {
```

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

```
2461 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2462 {
2463   \int_compare:nNnT
2464     { \box_ht:N \l_@@_cell_box }
2465     >
```

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).

```
2466   { \box_ht:N \strutbox }
2467   {
2468     \hbox_set:Nn \l_@@_cell_box
2469       {
2470         \box_move_down:nn
2471           {
2472             ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2473               + \baselineskip ) / 2
2474           }
2475         { \box_use:N \l_@@_cell_box }
2476       }
2477     }
2478   }
2479 }
```

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

```
2480 \cs_new:Npn \@@_V #1 #2
2481 {
2482   \str_if_eq:nnTF { #2 } { [ ] }
2483     { \@@_make_preamble_V_i:w [ ] }
2484     { \@@_make_preamble_V_i:w [ ] { #2 } }
2485 }
2486 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2487 { \@@_make_preamble_V_ii:nn { #1 } }
2488 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2489 {
2490   \str_set:Nn \l_@@_vpos_col_str { p }
2491   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2492   \@@_keys_p_column:n { #1 }
2493   \IfPackageLoadedTF { varwidth }
2494     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2495     {
2496       \@@_error_or_warning:n { varwidth-not-loaded }
2497       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2498     }
2499 }
```

For `w` and `W`

```
2500 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2501 \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.

```

2502 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2503 {
2504   \str_if_eq:nnTF { #3 } { s }
2505   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2506   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2507 }

```

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.

```

2508 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2509 {
2510   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2511   \tl_gclear:N \g_@@_pre_cell_tl
2512   \tl_gput_right:Nn \g_@@_array_preamble_tl
2513   {
2514     > {
2515       \dim_set:Nn \l_@@_col_width_dim { #2 }
2516       \@@_cell_begin:w
2517       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2518     }
2519     c
2520     < {
2521       \@@_cell_end_for_w_s:
2522       #1
2523       \@@_adjust_size_box:
2524       \box_use_drop:N \l_@@_cell_box
2525     }
2526   }
2527   \int_gincr:N \c@jCol
2528   \@@_rec_preamble_after_col:n
2529 }

```

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

```

2530 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2531 {
2532   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2533   \tl_gclear:N \g_@@_pre_cell_tl
2534   \tl_gput_right:Nn \g_@@_array_preamble_tl
2535   {
2536     > {

```

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.

```

2537       \dim_set:Nn \l_@@_col_width_dim { #4 }
2538       \hbox_set:Nw \l_@@_cell_box
2539       \@@_cell_begin:w
2540       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2541     }
2542     c
2543     < {
2544       \@@_cell_end:
2545       \hbox_set_end:
2546       #1
2547       \@@_adjust_size_box:
2548       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2549     }
2550   }

```

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

```

2551 \int_gincr:N \c@jCol
2552 \@@_rec_preamble_after_col:n
2553 }

2554 \cs_new_protected:Npn \@@_special_W:
2555 {
2556 \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2557 { \@@_warning:n { W~warning } }
2558 }

```

For S (of siunitx).

```

2559 \cs_new:Npn \@@_S #1 #2
2560 {
2561 \str_if_eq:nnTF { #2 } { [ ] }
2562 { \@@_make_preamble_S:w [ ] }
2563 { \@@_make_preamble_S:w [ ] { #2 } }
2564 }

2565 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2566 { \@@_make_preamble_S_i:n { #1 } }

2567 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2568 {
2569 \IfPackageLoadedTF { siunitx }
2570 {
2571 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2572 \tl_gclear:N \g_@@_pre_cell_tl
2573 \tl_gput_right:Nn \g_@@_array_preamble_tl
2574 {
2575 > {
2576 \@@_cell_begin:w
2577 \keys_set:nn { siunitx } { #1 }
2578 \siunitx_cell_begin:w
2579 }
2580 c
2581 < { \siunitx_cell_end: \@@_cell_end: }
2582 }

```

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

```

2583 \int_gincr:N \c@jCol
2584 \@@_rec_preamble_after_col:n
2585 }
2586 { \@@_fatal:n { siunitx~not~loaded } }
2587 }

```

For (, [and \{.

```

2588 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2589 {
2590 \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.

```

2591 \int_if_zero:nTF \c@jCol
2592 {
2593 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2594 {

```

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

```

2595 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2596 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2597 \@@_rec_preamble:n #2
2598 }
2599 {

```

```

2600         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2601         \@@_make_preamble_iv:nn { #1 } { #2 }
2602     }
2603 }
2604 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2605 }
2606 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2607 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2608 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2609 {
2610     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2611     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2612     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2613     {
2614         \@@_error:nn { delimiter~after~opening } { #2 }
2615         \@@_rec_preamble:n
2616     }
2617     { \@@_rec_preamble:n #2 }
2618 }

```

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

```

2619 \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2620 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2621 {
2622     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2623     \tl_if_in:nnTF { ) ] \} } { #2 }
2624     { \@@_make_preamble_v:nnn #1 #2 }
2625     {
2626         \tl_if_eq:nnTF { \stop } { #2 }
2627         {
2628             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2629             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2630             {
2631                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2632                 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2633                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2634                 \@@_rec_preamble:n #2
2635             }
2636         }
2637         {
2638             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2639             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2640             \tl_gput_right:Nx \g_@@_pre_code_after_tl
2641             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2642             \@@_rec_preamble:n #2
2643         }
2644     }
2645 }
2646 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2647 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2648 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2649 {
2650     \tl_if_eq:nnTF { \stop } { #3 }
2651     {
2652         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2653         {
2654             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2655 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2656 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2657 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2658 }
2659 {
2660 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2661 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2662 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2663 \@@_error:nn { double~closing~delimiter } { #2 }
2664 }
2665 }
2666 {
2667 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2668 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2669 \@@_error:nn { double~closing~delimiter } { #2 }
2670 \@@_rec_preamble:n #3
2671 }
2672 }

2673 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2674 { \use:c { @@ _ \token_to_str:N ) } }

```

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

```

2675 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2676 {
2677 \str_if_eq:nnTF { #1 } { < }
2678 \@@_rec_preamble_after_col_i:n
2679 {
2680 \str_if_eq:nnTF { #1 } { @ }
2681 \@@_rec_preamble_after_col_ii:n
2682 {
2683 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2684 {
2685 \tl_gput_right:Nn \g_@@_array_preamble_tl
2686 { ! { \skip_horizontal:N \arrayrulewidth } }
2687 }
2688 {
2689 \exp_args:NNe
2690 \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2691 {
2692 \tl_gput_right:Nn \g_@@_array_preamble_tl
2693 { ! { \skip_horizontal:N \arrayrulewidth } }
2694 }
2695 }
2696 \@@_rec_preamble:n { #1 }
2697 }
2698 }
2699 }

2700 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2701 {
2702 \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2703 \@@_rec_preamble_after_col:n
2704 }

```

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

```

2705 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2706 {
2707 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2708 {

```

```

2709     \tl_gput_right:Nn \g_@@_array_preamble_tl
2710     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2711   }
2712   {
2713     \exp_args:NNe
2714     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2715     {
2716       \tl_gput_right:Nn \g_@@_array_preamble_tl
2717       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2718     }
2719     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2720   }
2721   \@@_rec_preamble:n
2722 }

2723 \cs_new:cpn { @@ _ * } #1 #2 #3
2724 {
2725   \tl_clear:N \l_tmpa_tl
2726   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2727   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2728 }

```

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

```

2729 \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.

```

2730 \cs_new:Npn \@@_X #1 #2
2731 {
2732   \str_if_eq:nnTF { #2 } { [ ]
2733     { \@@_make_preamble_X:w [ ] }
2734     { \@@_make_preamble_X:w [ ] #2 }
2735   }
2736   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2737   { \@@_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 `{ nicematrix / 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`).

```

2738 \keys_define:nn { nicematrix / X-column }
2739 { 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.

```

2740 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2741 {

```

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).

```

2742   \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).

```

2743   \str_set:Nn \l_@@_vpos_col_str { p }

```

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

```

2744   \int_zero_new:N \l_@@_weight_int
2745   \int_set_eq:NN \l_@@_weight_int \c_one_int
2746   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in \l_tmpa_tl

```

2747 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2748 \int_compare:nNt \l_@@_weight_int < \c_zero_int
2749 {
2750   \@@_error_or_warning:n { negative-weight }
2751   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2752 }
2753 \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).

```

2754 \bool_if:NTF \l_@@_X_columns_aux_bool
2755 {
2756   \exp_args:Nne
2757   \@@_make_preamble_ii_iv:nnn
2758   { \l_@@_weight_int \l_@@_X_columns_dim }
2759   { minipage }
2760   { \@@_no_update_width: }
2761 }
2762 {
2763   \tl_gput_right:Nn \g_@@_array_preamble_tl
2764   {
2765     > {
2766       \@@_cell_begin:w
2767       \bool_set_true:N \l_@@_X_bool

```

You encounter a problem on 2023-03-04: for an environment with X columns, during the first compilations (which are not the definitive one), sometimes, some cells are declared empty even if they should not. That's a problem because user's instructions may use these nodes. That's why we have added the following \NotEmpty.

```

2768 \NotEmpty

```

The following code will nullify the box of the cell.

```

2769 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2770 { \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.

```

2771 \begin { minipage } { 5 cm } \arraybackslash
2772 }
2773 c
2774 < {
2775   \end { minipage }
2776   \@@_cell_end:
2777 }
2778 }
2779 \int_gincr:N \c_jCol
2780 \@@_rec_preamble_after_col:n
2781 }
2782 }

2783 \cs_new_protected:Npn \@@_no_update_width:
2784 {
2785   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2786   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2787 }

```

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

```

2788 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2789 {
2790   \seq_gput_right:Nx \g_@@_cols_vlism_seq
2791   { \int_eval:n { \c_jCol + 1 } }
2792   \tl_gput_right:Nx \g_@@_array_preamble_tl
2793   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }

```



```

2794 \@@_rec_preamble:n
2795 }

```

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.

```

2796 \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).

```

2797 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2798 { \@@_fatal:n { Preamble~forgotten } }
2799 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2800 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2801 \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).

```

2802 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2803 {

```

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

```

2804 \multispan { #1 }
2805 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2806 \begingroup
2807 \cs_set:Npn \@addamp
2808 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2809 \tl_gclear:N \g_@@_preamble_tl
2810 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2811 \exp_args:No \@mkpream \g_@@_preamble_tl
2812 \@addtopreamble \@empty
2813 \endgroup

```

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

```

2814 \int_compare:nNnT { #1 } > \c_one_int
2815 {
2816 \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2817 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2818 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2819 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2820 {
2821 {
2822 \int_if_zero:nTF \c@jCol
2823 { \int_eval:n { \c@iRow + 1 } }
2824 { \int_use:N \c@iRow }
2825 }
2826 { \int_eval:n { \c@jCol + 1 } }
2827 {
2828 \int_if_zero:nTF \c@jCol
2829 { \int_eval:n { \c@iRow + 1 } }
2830 { \int_use:N \c@iRow }

```

```

2831     }
2832     { \int_eval:n { \c@jCol + #1 } }
2833     { } % for the name of the block
2834 }
2835 }

```

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

```

2836 \cs_set:Npn \@sharp { #3 }
2837 \@arstrut
2838 \@preamble
2839 \null

```

We add some lines.

```

2840 \int_gadd:Nn \c@jCol { #1 - 1 }
2841 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2842   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2843 \ignorespaces
2844 }

```

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`.

```

2845 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2846 {
2847   \str_case:nnF { #1 }
2848   {
2849     c { \@@_make_m_preamble_i:n #1 }
2850     l { \@@_make_m_preamble_i:n #1 }
2851     r { \@@_make_m_preamble_i:n #1 }
2852     > { \@@_make_m_preamble_ii:nn #1 }
2853     ! { \@@_make_m_preamble_ii:nn #1 }
2854     @ { \@@_make_m_preamble_ii:nn #1 }
2855     | { \@@_make_m_preamble_iii:n #1 }
2856     p { \@@_make_m_preamble_iv:nnn t #1 }
2857     m { \@@_make_m_preamble_iv:nnn c #1 }
2858     b { \@@_make_m_preamble_iv:nnn b #1 }
2859     w { \@@_make_m_preamble_v:nnnn { } #1 }
2860     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2861     \q_stop { }
2862   }
2863   {
2864     \cs_if_exist:cTF { NC @ find @ #1 }
2865     {
2866       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2867       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2868     }
2869     {
2870       \tl_if_eq:nnT { #1 } { S }
2871       { \@@_fatal:n { unknown~column~type~S } }
2872       { \@@_fatal:nn { unknown~column~type } { #1 } }
2873     }
2874   }
2875 }

```

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

```

2876 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2877 {
2878   \tl_gput_right:Nn \g_@@_preamble_tl
2879   {
2880     > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2881     #1
2882     < \@@_cell_end:
2883   }

```

We test for the presence of a <.

```
2884 \@@_make_m_preamble_x:n
2885 }
```

For >, ! and @

```
2886 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2887 {
2888   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2889   \@@_make_m_preamble:n
2890 }
```

For |

```
2891 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2892 {
2893   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2894   \@@_make_m_preamble:n
2895 }
```

For p, m and b

```
2896 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2897 {
2898   \tl_gput_right:Nn \g_@@_preamble_tl
2899   {
2900     > {
2901       \@@_cell_begin:w
2902       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2903       \mode_leave_vertical:
2904       \arraybackslash
2905       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2906     }
2907     c
2908     < {
2909       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2910       \end { minipage }
2911       \@@_cell_end:
2912     }
2913   }
```

We test for the presence of a <.

```
2914 \@@_make_m_preamble_x:n
2915 }
```

For w and W

```
2916 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2917 {
2918   \tl_gput_right:Nn \g_@@_preamble_tl
2919   {
2920     > {
2921       \dim_set:Nn \l_@@_col_width_dim { #4 }
2922       \hbox_set:Nw \l_@@_cell_box
2923       \@@_cell_begin:w
2924       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2925     }
2926     c
2927     < {
2928       \@@_cell_end:
2929       \hbox_set_end:
2930       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2931       #1
2932       \@@_adjust_size_box:
2933       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2934     }
2935   }
```

We test for the presence of a <.

```
2936 \@@_make_m_preamble_x:n
2937 }
```

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

```
2938 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2939 {
2940   \str_if_eq:nnTF { #1 } { < }
2941     \@@_make_m_preamble_ix:n
2942     { \@@_make_m_preamble:n { #1 } }
2943 }
2944 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2945 {
2946   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2947   \@@_make_m_preamble_x:n
2948 }
```

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`).

```
2949 \cs_new_protected:Npn \@@_put_box_in_flow:
2950 {
2951   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2952   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2953   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2954     { \box_use_drop:N \l_tmpa_box }
2955   \@@_put_box_in_flow_i:
2956 }
```

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).

```
2957 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2958 {
2959   \pgfpicture
2960     \@@_qpoint:n { row - 1 }
2961     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2962     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2963     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2964     \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).

```
2965   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2966   {
2967     \int_set:Nn \l_tmpa_int
2968     {
2969       \str_range:Nnn
2970         \l_@@_baseline_tl
2971         6
2972         { \tl_count:o \l_@@_baseline_tl }
2973     }
2974     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2975   }
2976   {
2977     \tl_if_eq:NnTF \l_@@_baseline_tl { t }
2978     { \int_set_eq:NN \l_tmpa_int \c_one_int }
2979     {
2980       \tl_if_eq:NnTF \l_@@_baseline_tl { b }
2981       { \int_set_eq:NN \l_tmpa_int \c@iRow }
2982       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
```

```

2983     }
2984     \bool_lazy_or:nnT
2985     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2986     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2987     {
2988         \@@_error:n { bad-value-for-baseline }
2989         \int_set_eq:NN \l_tmpa_int \c_one_int
2990     }
2991     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2992     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2993     }
2994     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2995     \endpgfpicture
2996     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2997     \box_use_drop:N \l_tmpa_box
2998 }

```

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).

```

2999 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3000 {

```

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.

```

3001     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3002     {
3003         \int_compare:nNnT \c_jCol > \c_one_int
3004         {
3005             \box_set_wd:Nn \l_@@_the_array_box
3006             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3007         }
3008     }

```

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

```

3009     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3010     \bool_if:NT \l_@@_caption_above_bool
3011     {
3012         \tl_if_empty:NF \l_@@_caption_tl
3013         {
3014             \bool_set_false:N \g_@@_caption_finished_bool
3015             \int_gzero:N \c@tabularnote
3016             \@@_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).

```

3017         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3018         {
3019             \tl_gput_right:Nx \g_@@_aux_tl
3020             {
3021                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3022                 { \int_use:N \g_@@_notes_caption_int }
3023             }
3024             \int_gzero:N \g_@@_notes_caption_int
3025         }
3026     }
3027 }

```

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

```

3028   \hbox
3029   {
3030   \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.

```

3031   \@@_create_extra_nodes:
3032   \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3033 }

```

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

```

3034   \bool_lazy_any:nT
3035   {
3036     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3037     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3038     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3039   }
3040   \@@_insert_tabularnotes:
3041   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3042   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3043   \end { minipage }
3044 }

```

```

3045 \cs_new_protected:Npn \@@_insert_caption:
3046 {
3047   \tl_if_empty:NF \l_@@_caption_tl
3048   {
3049     \cs_if_exist:NTF \capttype
3050     { \@@_insert_caption_i: }
3051     { \@@_error:n { caption~outside~float } }
3052   }
3053 }

```

```

3054 \cs_new_protected:Npn \@@_insert_caption_i:
3055 {
3056   \group_begin:

```

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

```

3057   \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.

```

3058   \IfPackageLoadedTF { floatrow }
3059   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3060   { }
3061   \tl_if_empty:NTF \l_@@_short_caption_tl
3062   { \caption }
3063   { \caption [ \l_@@_short_caption_tl ] }
3064   { \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.

```

3065 \bool_if:NF \g_@@_caption_finished_bool
3066 {
3067   \bool_gset_true:N \g_@@_caption_finished_bool
3068   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabulernote
3069   \int_gzero:N \c@tabulernote
3070 }
3071 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3072 \group_end:
3073 }
3074 \cs_new_protected:Npn \@@_tabulernote_error:n #1
3075 {
3076   \@@_error_or_warning:n { tabulernote~below~the~tabular }
3077   \@@_gredirect_none:n { tabulernote~below~the~tabular }
3078 }
3079 \cs_new_protected:Npn \@@_insert_tabularnotes:
3080 {
3081   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3082   \int_set:Nn \c@tabulernote { \seq_count:N \g_@@_notes_seq }
3083   \skip_vertical:N 0.65ex

```

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

```

3084 \group_begin:
3085 \l_@@_notes_code_before_tl
3086 \tl_if_empty:NF \g_@@_tabulernote_tl
3087 {
3088   \g_@@_tabulernote_tl \par
3089   \tl_gclear:N \g_@@_tabulernote_tl
3090 }

```

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.

```

3091 \int_compare:nNnT \c@tabulernote > \c_zero_int
3092 {
3093   \bool_if:NTF \l_@@_notes_para_bool
3094   {
3095     \begin { tabularnotes* }
3096     \seq_map_inline:Nn \g_@@_notes_seq
3097       { \@@_one_tabulernote:nn ##1 }
3098     \strut
3099     \end { tabularnotes* }

```

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

```

3100 \par
3101 }
3102 {
3103   \tabularnotes
3104   \seq_map_inline:Nn \g_@@_notes_seq
3105     { \@@_one_tabulernote:nn ##1 }
3106   \strut
3107   \endtabularnotes
3108 }
3109 }
3110 \unskip
3111 \group_end:
3112 \bool_if:NT \l_@@_notes_bottomrule_bool
3113 {
3114   \IfPackageLoadedTF { booktabs }
3115   {

```

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

```

3116 \skip_vertical:N \aboverulesep

```

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

```

3117         { \CT@arc@ \hrule height \heavyrulewidth }
3118     }
3119     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3120 }
3121 \l_@@_notes_code_after_tl
3122 \seq_gclear:N \g_@@_notes_seq
3123 \seq_gclear:N \g_@@_notes_in_caption_seq
3124 \int_gzero:N \c@tabularnote
3125 }

```

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.

```

3126 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3127 {
3128     \tl_if_novalue:nTF { #1 }
3129     { \item }
3130     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3131 }

```

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`.

```

3132 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3133 {
3134     \pgfpicture
3135     \@@_qpoint:n { row - 1 }
3136     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3137     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3138     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3139     \endpgfpicture
3140     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3141     \int_if_zero:nT \l_@@_first_row_int
3142     {
3143         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3144         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3145     }
3146     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3147 }

```

Now, the general case.

```

3148 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3149 {

```

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

```

3150     \tl_if_eq:NnT \l_@@_baseline_tl { t }
3151     { \cs_set_nopar:Npn \l_@@_baseline_tl { 1 } }

```

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

```

3152     \pgfpicture
3153     \@@_qpoint:n { row - 1 }
3154     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3155     \str_if_in:NnTF \l_@@_baseline_tl { line- }
3156     {
3157         \int_set:Nn \l_tmpa_int
3158         {
3159             \str_range:Nnn
3160             \l_@@_baseline_tl
3161             6
3162             { \tl_count:o \l_@@_baseline_tl }

```



```

3163     }
3164     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3165   }
3166   {
3167     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3168     \bool_lazy_or:nnT
3169       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3170       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3171     {
3172       \@@_error:n { bad-value-for-baseline }
3173       \int_set:Nn \l_tmpa_int 1
3174     }
3175     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3176   }
3177   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3178   \endpgfpicture
3179   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3180   \int_if_zero:nT \l_@@_first_row_int
3181   {
3182     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3183     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3184   }
3185   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3186 }

```

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.

```

3187 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3188 {

```

We will compute the real width of both delimiters used.

```

3189   \dim_zero_new:N \l_@@_real_left_delim_dim
3190   \dim_zero_new:N \l_@@_real_right_delim_dim
3191   \hbox_set:Nn \l_tmpb_box
3192   {
3193     \c_math_toggle_token
3194     \left #1
3195     \vcenter
3196     {
3197       \vbox_to_ht:nn
3198         { \box_ht_plus_dp:N \l_tmpa_box }
3199         { }
3200     }
3201     \right .
3202     \c_math_toggle_token
3203   }
3204   \dim_set:Nn \l_@@_real_left_delim_dim
3205   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3206   \hbox_set:Nn \l_tmpb_box
3207   {
3208     \c_math_toggle_token
3209     \left .
3210     \vbox_to_ht:nn
3211       { \box_ht_plus_dp:N \l_tmpa_box }
3212       { }
3213     \right #2
3214     \c_math_toggle_token
3215   }
3216   \dim_set:Nn \l_@@_real_right_delim_dim
3217   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3218   \skip_horizontal:N \l_@@_left_delim_dim

```

```

3219 \skip_horizontal:N -\l_@@_real_left_delim_dim
3220 \@@_put_box_in_flow:
3221 \skip_horizontal:N \l_@@_right_delim_dim
3222 \skip_horizontal:N -\l_@@_real_right_delim_dim
3223 }

```

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).

```

3224 \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).

```

3225 {
3226   \peek_remove_spaces:n
3227   {
3228     \peek_meaning:NTF \end
3229     \@@_analyze_end:Nn
3230     {
3231       \@@_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`).

```

3232       \exp_args:No \@@_array: \g_@@_array_preamble_tl
3233     }
3234   }
3235 }
3236 {
3237   \@@_create_col_nodes:
3238   \endarray
3239 }

```

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

```

3240 \NewDocumentEnvironment { @@-light-syntax } { b }
3241 {

```

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`.

```

3242   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3243   \tl_map_inline:nn { #1 }
3244   {
3245     \str_if_eq:nnT { ##1 } { & }
3246     { \@@_fatal:n { ampersand-in~light-syntax } }
3247     \str_if_eq:nnT { ##1 } { \ }
3248     { \@@_fatal:n { double-backslash-in~light-syntax } }
3249   }

```

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`.

```

3250   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3251 }

```

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.

```

3252 {
3253   \@@_create_col_nodes:
3254   \endarray
3255 }

3256 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3257 {
3258   \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).

```

3259   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3260   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3261   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3262     \seq_set_split:Nee
3263     \seq_set_split:NVn
3264     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3265   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3266   \tl_if_empty:NF \l_tmpa_tl
3267   { \seq_put_right:No \l_@@_rows_seq \l_tmpa_tl }

```

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

```

3268   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3269     { \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`).

```

3270   \tl_build_begin:N \l_@@_new_body_tl
3271   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3272   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3273   \@@_line_with_light_syntax:o \l_tmpa_tl

```

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

```

3274   \seq_map_inline:Nn \l_@@_rows_seq
3275   {
3276     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3277     \@@_line_with_light_syntax:n { ##1 }
3278   }
3279   \tl_build_end:N \l_@@_new_body_tl

3280   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3281   {
3282     \int_set:Nn \l_@@_last_col_int
3283     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3284   }

```

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.

```

3285   \@@_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`).

```

3286   \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3287 }

```

```

3288 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3289 {
3290   \seq_clear_new:N \l_@@_cells_seq
3291   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3292   \int_set:Nn \l_@@_nb_cols_int
3293   {
3294     \int_max:nn
3295     \l_@@_nb_cols_int
3296     { \seq_count:N \l_@@_cells_seq }
3297   }
3298   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3299   \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3300   \seq_map_inline:Nn \l_@@_cells_seq
3301   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3302 }
3303 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }

```

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

```

3304 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3305 {
3306   \str_if_eq:onT \g_@@_name_env_str { #2 }
3307   { \@@_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.

```

3308   \end { #2 }
3309 }

```

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).

```

3310 \cs_new:Npn \@@_create_col_nodes:
3311 {
3312   \crrr
3313   \int_if_zero:nT \l_@@_first_col_int
3314   {
3315     \omit
3316     \hbox_overlap_left:n
3317     {
3318       \bool_if:NT \l_@@_code_before_bool
3319       { \pgfsys@markposition { \@@_env: - col - 0 } }
3320       \pgfpicture
3321       \pgfrememberpicturepositiononpagetrue
3322       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3323       \str_if_empty:NF \l_@@_name_str
3324       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3325       \endpgfpicture
3326       \skip_horizontal:N 2\col@sep
3327       \skip_horizontal:N \g_@@_width_first_col_dim
3328     }
3329     &
3330   }
3331   \omit

```

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

```

3332   \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).

```

3333   \int_if_zero:nTF \l_@@_first_col_int
3334   {

```

```

3335 \bool_if:NT \l_@@_code_before_bool
3336 {
3337   \hbox
3338   {
3339     \skip_horizontal:N -0.5\arrayrulewidth
3340     \pgfsys@markposition { \@@_env: - col - 1 }
3341     \skip_horizontal:N 0.5\arrayrulewidth
3342   }
3343 }
3344 \pgfpicture
3345 \pgfrememberpicturepositiononpagetrue
3346 \pgfcoordinate { \@@_env: - col - 1 }
3347 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3348 \str_if_empty:NF \l_@@_name_str
3349 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3350 \endpgfpicture
3351 }
3352 {
3353 \bool_if:NT \l_@@_code_before_bool
3354 {
3355   \hbox
3356   {
3357     \skip_horizontal:N 0.5\arrayrulewidth
3358     \pgfsys@markposition { \@@_env: - col - 1 }
3359     \skip_horizontal:N -0.5\arrayrulewidth
3360   }
3361 }
3362 \pgfpicture
3363 \pgfrememberpicturepositiononpagetrue
3364 \pgfcoordinate { \@@_env: - col - 1 }
3365 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3366 \str_if_empty:NF \l_@@_name_str
3367 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3368 \endpgfpicture
3369 }

```

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

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

```

3370 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3371 \bool_if:NF \l_@@_auto_columns_width_bool
3372 { \dim_compare:nNt \l_@@_columns_width_dim > \c_zero_dim }
3373 {
3374   \bool_lazy_and:nnTF
3375   \l_@@_auto_columns_width_bool
3376   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3377   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3378   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3379   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3380 }
3381 \skip_horizontal:N \g_tmpa_skip
3382 \hbox
3383 {
3384 \bool_if:NT \l_@@_code_before_bool
3385 {
3386   \hbox
3387   {
3388     \skip_horizontal:N -0.5\arrayrulewidth
3389     \pgfsys@markposition { \@@_env: - col - 2 }
3390     \skip_horizontal:N 0.5\arrayrulewidth

```

```

3391     }
3392   }
3393   \pgfpicture
3394   \pgfrememberpicturepositiononpagetrue
3395   \pgfcoordinate { \@@_env: - col - 2 }
3396   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3397   \str_if_empty:NF \l_@@_name_str
3398   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3399   \endpgfpicture
3400 }

```

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.

```

3401   \int_gset_eq:NN \g_tmpa_int \c_one_int
3402   \bool_if:NTF \g_@@_last_col_found_bool
3403   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3404   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3405   {
3406     &
3407     \omit
3408     \int_gincr:N \g_tmpa_int

```

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

```

3409     \skip_horizontal:N \g_tmpa_skip
3410     \bool_if:NT \l_@@_code_before_bool
3411     {
3412       \hbox
3413       {
3414         \skip_horizontal:N -0.5\arrayrulewidth
3415         \pgfsys@markposition
3416         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3417         \skip_horizontal:N 0.5\arrayrulewidth
3418       }
3419     }

```

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

```

3420   \pgfpicture
3421   \pgfrememberpicturepositiononpagetrue
3422   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3423   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3424   \str_if_empty:NF \l_@@_name_str
3425   {
3426     \pgfnodealias
3427     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3428     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3429   }
3430   \endpgfpicture
3431 }

3432 &
3433 \omit

```

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

```

3434   \int_if_zero:nT \g_@@_col_total_int
3435   { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3436   \skip_horizontal:N \g_tmpa_skip
3437   \int_gincr:N \g_tmpa_int
3438   \bool_lazy_any:nF % modified 2023/12/13
3439   {
3440     \g_@@_delims_bool
3441     \l_@@_tabular_bool
3442     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3443     \l_@@_exterior_arraycolsep_bool

```

```

3444         \l_@@_bar_at_end_of_pream_bool
3445     }
3446     { \skip_horizontal:N -\col@sep }
3447 \bool_if:NT \l_@@_code_before_bool
3448 {
3449     \hbox
3450     {
3451         \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.

```

3452         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3453         { \skip_horizontal:N -\arraycolsep }
3454         \pgfsys@markposition
3455         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3456         \skip_horizontal:N 0.5\arrayrulewidth
3457         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3458         { \skip_horizontal:N \arraycolsep }
3459     }
3460 }
3461 \pgfpicture
3462 \pgfrememberpicturerepositiononpagetrue
3463 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3464 {
3465     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3466     {
3467         \pgfpoint
3468         { - 0.5 \arrayrulewidth - \arraycolsep }
3469         \c_zero_dim
3470     }
3471     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3472 }
3473 \str_if_empty:NF \l_@@_name_str
3474 {
3475     \pgfnodealias
3476     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3477     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3478 }
3479 \endpgfpicture

3480 \bool_if:NT \g_@@_last_col_found_bool
3481 {
3482     \hbox_overlap_right:n
3483     {
3484         \skip_horizontal:N \g_@@_width_last_col_dim
3485         \skip_horizontal:N \col@sep
3486         \bool_if:NT \l_@@_code_before_bool
3487         {
3488             \pgfsys@markposition
3489             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3490         }
3491         \pgfpicture
3492         \pgfrememberpicturerepositiononpagetrue
3493         \pgfcoordinate
3494         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3495         \pgfpintorigin
3496         \str_if_empty:NF \l_@@_name_str
3497         {
3498             \pgfnodealias
3499             {
3500                 \l_@@_name_str - col
3501                 - \int_eval:n { \g_@@_col_total_int + 1 }

```

```

3502         }
3503         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3504     }
3505     \endpgfpicture
3506 }
3507 }
3508 \cr
3509 }

```

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

```

3510 \tl_const:Nn \c_@@_preamble_first_col_tl
3511 {
3512     >
3513     {

```

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).

```

3514         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3515         \bool_gset_true:N \g_@@_after_col_zero_bool
3516         \@@_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.

```

3517         \hbox_set:Nw \l_@@_cell_box
3518         \@@_math_toggle:
3519         \@@_tuning_key_small:

```

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

```

3520         \int_compare:nNnT \c@iRow > \c_zero_int
3521         {
3522             \bool_lazy_or:nnT
3523             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3524             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3525             {
3526                 \l_@@_code_for_first_col_tl
3527                 \xglobal \colorlet { nicematrix-first-col } { . }
3528             }
3529         }
3530     }

```

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`.

```

3531     l
3532     <
3533     {
3534         \@@_math_toggle:
3535         \hbox_set_end:
3536         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3537         \@@_adjust_size_box:
3538         \@@_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.

```

3539         \dim_gset:Nn \g_@@_width_first_col_dim
3540         { \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.

```

3541         \hbox_overlap_left:n
3542         {
3543             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3544             \@@_node_for_cell:
3545             { \box_use_drop:N \l_@@_cell_box }
3546             \skip_horizontal:N \l_@@_left_delim_dim

```



```

3547         \skip_horizontal:N \l_@@_left_margin_dim
3548         \skip_horizontal:N \l_@@_extra_left_margin_dim
3549     }
3550     \bool_gset_false:N \g_@@_empty_cell_bool
3551     \skip_horizontal:N -2\col@sep
3552 }
3553 }

```

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

```

3554 \tl_const:Nn \c_@@_preamble_last_col_tl
3555 {
3556     >
3557     {
3558         \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).

```

3559         \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.

```

3560         \bool_gset_true:N \g_@@_last_col_found_bool
3561         \int_gincr:N \c@jCol
3562         \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.

```

3563         \hbox_set:Nw \l_@@_cell_box
3564         \@@_math_toggle:
3565         \@@_tuning_key_small:

```

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

```

3566         \int_compare:nNnT \c@iRow > \c_zero_int
3567         {
3568             \bool_lazy_or:nnT
3569             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3570             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3571             {
3572                 \l_@@_code_for_last_col_tl
3573                 \xglobal \colorlet { nicematrix-last-col } { . }
3574             }
3575         }
3576     }
3577     1
3578     <
3579     {
3580         \@@_math_toggle:
3581         \hbox_set_end:
3582         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3583         \@@_adjust_size_box:
3584         \@@_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.

```

3585         \dim_gset:Nn \g_@@_width_last_col_dim
3586         { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3587         \skip_horizontal:N -2\col@sep

```

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

```

3588         \hbox_overlap_right:n
3589         {
3590             \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3591             {
3592                 \skip_horizontal:N \l_@@_right_delim_dim
3593                 \skip_horizontal:N \l_@@_right_margin_dim
3594                 \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

3595         \@@_node_for_cell:
3596     }
3597 }
3598 \bool_gset_false:N \g_@@_empty_cell_bool
3599 }
3600 }

```

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

```

3601 \NewDocumentEnvironment { NiceArray } { }
3602 {
3603     \bool_gset_false:N \g_@@_delims_bool
3604     \str_if_empty:NT \g_@@_name_env_str
3605     { \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).

```

3606     \NiceArrayWithDelims . .
3607 }
3608 { \endNiceArrayWithDelims }

```

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

```

3609 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3610 {
3611     \NewDocumentEnvironment { #1 NiceArray } { }
3612     {
3613         \bool_gset_true:N \g_@@_delims_bool
3614         \str_if_empty:NT \g_@@_name_env_str
3615         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3616         \@@_test_if_math_mode:
3617         \NiceArrayWithDelims #2 #3
3618     }
3619     { \endNiceArrayWithDelims }
3620 }
3621 \@@_def_env:nnn p ( )
3622 \@@_def_env:nnn b [ ]
3623 \@@_def_env:nnn B \{ \}
3624 \@@_def_env:nnn v | |
3625 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3626 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3627 {
3628     \bool_set_false:N \l_@@_preamble_bool
3629     \tl_clear:N \l_tmpa_tl
3630     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3631     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3632     \tl_put_right:Nn \l_tmpa_tl
3633     {
3634         *
3635         {
3636             \int_case:nnF \l_@@_last_col_int
3637             {
3638                 { -2 } { \c@MaxMatrixCols }
3639                 { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3640         }
3641         { \int_eval:n { \l_@@_last_col_int - 1 } }
3642     }
3643     { #2 }
3644 }
3645 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3646 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3647 }
3648 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3649 \clist_map_inline:nn { p , b , B , v , V }
3650 {
3651     \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3652     {
3653         \bool_gset_true:N \g_@@_delims_bool
3654         \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3655         \int_if_zero:nT \l_@@_last_col_int
3656         {
3657             \bool_set_true:N \l_@@_last_col_without_value_bool
3658             \int_set:Nn \l_@@_last_col_int { -1 }
3659         }
3660         \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3661         \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3662     }
3663     { \use:c { end #1 NiceArray } }
3664 }

```

We define also an environment {NiceMatrix}

```

3665 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3666 {
3667     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3668     \int_if_zero:nT \l_@@_last_col_int
3669     {
3670         \bool_set_true:N \l_@@_last_col_without_value_bool
3671         \int_set:Nn \l_@@_last_col_int { -1 }
3672     }
3673     \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3674     \bool_lazy_or:nnT
3675     { \clist_if_empty_p:N \l_@@_vlines_clist }
3676     { \l_@@_except_borders_bool }
3677     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3678     \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3679 }
3680 { \endNiceArray }

```

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

```

3681 \cs_new_protected:Npn \@@_NotEmpty:
3682 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3683 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3684 {

```

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.

```

3685     \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3686     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3687     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3688     \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3689     \tl_if_empty:NF \l_@@_short_caption_tl
3690     {

```

```

3691     \tl_if_empty:NT \l_@@_caption_tl
3692     {
3693         \@@_error_or_warning:n { short-caption-without~caption }
3694         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3695     }
3696 }
3697 \tl_if_empty:NF \l_@@_label_tl
3698 {
3699     \tl_if_empty:NT \l_@@_caption_tl
3700     { \@@_error_or_warning:n { label~without~caption } }
3701 }
3702 \NewDocumentEnvironment { TabularNote } { b }
3703 {
3704     \bool_if:NTF \l_@@_in_code_after_bool
3705     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3706     {
3707         \tl_if_empty:NF \g_@@_tabularnote_tl
3708         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3709         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3710     }
3711 }
3712 { }
3713 \@@_settings_for_tabular:
3714 \NiceArray { #2 }
3715 }
3716 { \endNiceArray }
3717 \cs_new_protected:Npn \@@_settings_for_tabular:
3718 {
3719     \bool_set_true:N \l_@@_tabular_bool
3720     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3721     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3722     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3723 }

3724 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3725 {
3726     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3727     \dim_zero_new:N \l_@@_width_dim
3728     \dim_set:Nn \l_@@_width_dim { #1 }
3729     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3730     \@@_settings_for_tabular:
3731     \NiceArray { #3 }
3732 }
3733 {
3734     \endNiceArray
3735     \int_if_zero:nT \g_@@_total_X_weight_int
3736     { \@@_error:n { NiceTabularX~without~X } }
3737 }

3738 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3739 {
3740     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3741     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3742     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3743     \@@_settings_for_tabular:
3744     \NiceArray { #3 }
3745 }
3746 { \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`).

```

3747 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3748 {
3749   \bool_lazy_all:nT
3750   {
3751     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3752     \l_@@_hvlines_bool
3753     { ! \g_@@_delims_bool }
3754     { ! \l_@@_except_borders_bool }
3755   }
3756   {
3757     \bool_set_true:N \l_@@_except_borders_bool
3758     \clist_if_empty:NF \l_@@_corners_clist
3759     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3760     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3761     {
3762       \@@_stroke_block:nnn
3763       {
3764         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3765         draw = \l_@@_rules_color_tl
3766       }
3767       { 1-1 }
3768       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3769     }
3770   }
3771 }

3772 \cs_new_protected:Npn \@@_after_array:
3773 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_ii:` in order to come back to the standard definition of `\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```

3774 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3775 \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.

```

3776 \bool_if:NT \g_@@_last_col_found_bool
3777 { \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`.

```

3778 \bool_if:NT \l_@@_last_col_without_value_bool
3779 { \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.

```

3780 \bool_if:NT \l_@@_last_row_without_value_bool
3781 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3782 \tl_gput_right:Nx \g_@@_aux_tl
3783 {
3784   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3785   {
3786     \int_use:N \l_@@_first_row_int ,
3787     \int_use:N \c@iRow ,
3788     \int_use:N \g_@@_row_total_int ,
3789     \int_use:N \l_@@_first_col_int ,
3790     \int_use:N \c@jCol ,
3791     \int_use:N \g_@@_col_total_int
3792   }
3793 }

```

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`).

```

3794 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3795 {
3796   \tl_gput_right:Nx \g_@@_aux_tl
3797   {
3798     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3799     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3800   }
3801 }
3802 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3803 {
3804   \tl_gput_right:Nx \g_@@_aux_tl
3805   {
3806     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3807     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3808     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3809     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3810   }
3811 }

```

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

```

3812 \@@_create_diag_nodes:

```

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

```

3813 \pgfpicture
3814 \int_step_inline:nn \c@iRow
3815 {
3816   \pgfnodealias
3817   { \@@_env: - ##1 - last }
3818   { \@@_env: - ##1 - \int_use:N \c@jCol }
3819 }
3820 \int_step_inline:nn \c@jCol
3821 {
3822   \pgfnodealias
3823   { \@@_env: - last - ##1 }
3824   { \@@_env: - \int_use:N \c@iRow - ##1 }
3825 }
3826 \str_if_empty:NF \l_@@_name_str
3827 {
3828   \int_step_inline:nn \c@iRow
3829   {
3830     \pgfnodealias
3831     { \l_@@_name_str - ##1 - last }
3832     { \@@_env: - ##1 - \int_use:N \c@jCol }
3833   }
3834   \int_step_inline:nn \c@jCol
3835   {
3836     \pgfnodealias

```

```

3837         { \l_@@_name_str - last - ##1 }
3838         { \@@_env: - \int_use:N \c@iRow - ##1 }
3839     }
3840 }
3841 \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.

```

3842 \bool_if:NT \l_@@_parallelize_diags_bool
3843 {
3844     \int_gzero_new:N \g_@@_ddots_int
3845     \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.

```

3846     \dim_gzero_new:N \g_@@_delta_x_one_dim
3847     \dim_gzero_new:N \g_@@_delta_y_one_dim
3848     \dim_gzero_new:N \g_@@_delta_x_two_dim
3849     \dim_gzero_new:N \g_@@_delta_y_two_dim
3850 }
3851 \int_zero_new:N \l_@@_initial_i_int
3852 \int_zero_new:N \l_@@_initial_j_int
3853 \int_zero_new:N \l_@@_final_i_int
3854 \int_zero_new:N \l_@@_final_j_int
3855 \bool_set_false:N \l_@@_initial_open_bool
3856 \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.

```

3857 \bool_if:NT \l_@@_small_bool
3858 {
3859     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3860     \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.

```

3861     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3862     { 0.6 \l_@@_xdots_shorten_start_dim }
3863     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3864     { 0.6 \l_@@_xdots_shorten_end_dim }
3865 }

```

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

```

3866 \@@_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.

```

3867 \@@_compute_corners:

```

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

```

3868 \@@_adjust_pos_of_blocks_seq:

```

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

```

3869 \@@_deal_with_rounded_corners:
3870 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3871 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3872 \IfPackageLoadedTF { tikz }
3873 {
3874   \tikzset
3875   {
3876     every~picture / .style =
3877     {
3878       overlay ,
3879       remember~picture ,
3880       name~prefix = \@@_env: -
3881     }
3882   }
3883 }
3884 { }
3885 \bool_if:NT \c_@@_tagging_array_bool
3886 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3887 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3888 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3889 \cs_set_eq:NN \OverBrace \@@_OverBrace
3890 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3891 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3892 \cs_set_eq:NN \line \@@_line
3893 \g_@@_pre_code_after_tl
3894 \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.

```

3895 \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).

```

3896 \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`).

```

3897 % \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3898 % { \@@_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:`.

```

3899 \bool_set_true:N \l_@@_in_code_after_bool
3900 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3901 \scan_stop:
3902 \tl_gclear:N \g_nicematrix_code_after_tl
3903 \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.

```

3904 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3905 \tl_if_empty:NF \g_@@_pre_code_before_tl
3906 {
3907   \tl_gput_right:Nx \g_@@_aux_tl
3908   {
3909     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl

```



```

3910         { \exp_not:o \g_@@_pre_code_before_tl }
3911     }
3912     \tl_gclear:N \g_@@_pre_code_before_tl
3913 }
3914 \tl_if_empty:NF \g_nicematrix_code_before_tl
3915 {
3916     \tl_gput_right:Nx \g_@@_aux_tl
3917     {
3918         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3919         { \exp_not:o \g_nicematrix_code_before_tl }
3920     }
3921     \tl_gclear:N \g_nicematrix_code_before_tl
3922 }

3923 \str_gclear:N \g_@@_name_env_str
3924 \@@_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`.

```

3925     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3926 }

```

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`.

```

3927 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3928 { \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).

```

3929 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3930 {
3931     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3932     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3933 }

```

The following command must *not* be protected.

```

3934 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3935 {
3936     { #1 }
3937     { #2 }
3938     {
3939         \int_compare:nNnTF { #3 } > { 99 }
3940         { \int_use:N \c@iRow }
3941         { #3 }
3942     }
3943     {
3944         \int_compare:nNnTF { #4 } > { 99 }
3945         { \int_use:N \c@jCol }
3946         { #4 }
3947     }
3948     { #5 }
3949 }

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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).

```

3950 \hook_gput_code:nnn { begindocument } { . }
3951 {
3952   \cs_new_protected:Npx \@@_draw_dotted_lines:
3953   {
3954     \c_@@_pgfortikzpicture_tl
3955     \@@_draw_dotted_lines_i:
3956     \c_@@_endpgfortikzpicture_tl
3957   }
3958 }

```

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

```

3959 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3960 {
3961   \pgfrememberpicturepositiononpagetrue
3962   \pgf@relevantforpicturesizefalse
3963   \g_@@_HVdotsfor_lines_tl
3964   \g_@@_Vdots_lines_tl
3965   \g_@@_Ddots_lines_tl
3966   \g_@@_Idots_lines_tl
3967   \g_@@_Cdots_lines_tl
3968   \g_@@_Ldots_lines_tl
3969 }

3970 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3971 {
3972   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3973   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3974 }

```

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

```

3975 \pgfdeclareshape { @@_diag_node }
3976 {
3977   \savedanchor { \five }
3978   {
3979     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3980     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3981   }
3982   \anchor { 5 } { \five }
3983   \anchor { center } { \pgfpointorigin }
3984 }

```

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).

```

3985 \cs_new_protected:Npn \@@_create_diag_nodes:
3986 {
3987   \pgfpicture
3988   \pgfrememberpicturepositiononpagetrue
3989   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3990   {
3991     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3992     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3993     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3994     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3995     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3996     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3997     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }

```

```

3998      \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3999      \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.

```

4000      \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4001      \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4002      \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4003      \str_if_empty:NF \l_@@_name_str
4004      { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4005  }

```

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

```

4006      \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4007      \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4008      \dim_set_eq:NN \l_tmpa_dim \pgf@y
4009      \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4010      \pgfcoordinate
4011      { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4012      \pgfnodealias
4013      { \@@_env: - last }
4014      { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4015      \str_if_empty:NF \l_@@_name_str
4016      {
4017          \pgfnodealias
4018          { \l_@@_name_str - \int_use:N \l_tmpa_int }
4019          { \@@_env: - \int_use:N \l_tmpa_int }
4020          \pgfnodealias
4021          { \l_@@_name_str - last }
4022          { \@@_env: - last }
4023      }
4024      \endpgfpicture
4025  }

```

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.

```
4026 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4027 {
```

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

```
4028 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4029 \int_set:Nn \l_@@_initial_i_int { #1 }
4030 \int_set:Nn \l_@@_initial_j_int { #2 }
4031 \int_set:Nn \l_@@_final_i_int { #1 }
4032 \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.

```
4033 \bool_set_false:N \l_@@_stop_loop_bool
4034 \bool_do_until:Nn \l_@@_stop_loop_bool
4035 {
4036   \int_add:Nn \l_@@_final_i_int { #3 }
4037   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4038   \bool_set_false:N \l_@@_final_open_bool
4039   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4040   {
4041     \int_compare:nNnTF { #3 } = \c_one_int
4042     { \bool_set_true:N \l_@@_final_open_bool }
4043     {
4044       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4045       { \bool_set_true:N \l_@@_final_open_bool }
4046     }
4047   }
4048   {
4049     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
4050     {
4051       \int_compare:nNnT { #4 } = { -1 }
4052       { \bool_set_true:N \l_@@_final_open_bool }
4053     }
4054     {
4055       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4056       {
4057         \int_compare:nNnT { #4 } = \c_one_int
4058         { \bool_set_true:N \l_@@_final_open_bool }
4059       }
4060     }
4061   }
4062   \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.

```
4063   {
```

We do a step backwards.

```
4064     \int_sub:Nn \l_@@_final_i_int { #3 }
4065     \int_sub:Nn \l_@@_final_j_int { #4 }
4066     \bool_set_true:N \l_@@_stop_loop_bool
4067   }
```

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`.

```

4068     {
4069         \cs_if_exist:cTF
4070         {
4071             @@ _ dotted _
4072             \int_use:N \l_@@_final_i_int -
4073             \int_use:N \l_@@_final_j_int
4074         }
4075         {
4076             \int_sub:Nn \l_@@_final_i_int { #3 }
4077             \int_sub:Nn \l_@@_final_j_int { #4 }
4078             \bool_set_true:N \l_@@_final_open_bool
4079             \bool_set_true:N \l_@@_stop_loop_bool
4080         }
4081     }
4082     \cs_if_exist:cTF
4083     {
4084         pgf @ sh @ ns @ \@@_env:
4085         - \int_use:N \l_@@_final_i_int
4086         - \int_use:N \l_@@_final_j_int
4087     }
4088     { \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.

```

4089     {
4090         \cs_set:cpn
4091         {
4092             @@ _ dotted _
4093             \int_use:N \l_@@_final_i_int -
4094             \int_use:N \l_@@_final_j_int
4095         }
4096         { }
4097     }
4098 }
4099 }
4100 }

```

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

```

4101     \bool_set_false:N \l_@@_stop_loop_bool
4102     \bool_do_until:Nn \l_@@_stop_loop_bool
4103     {
4104         \int_sub:Nn \l_@@_initial_i_int { #3 }
4105         \int_sub:Nn \l_@@_initial_j_int { #4 }
4106         \bool_set_false:N \l_@@_initial_open_bool
4107         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4108         {
4109             \int_compare:nNnTF { #3 } = \c_one_int
4110             { \bool_set_true:N \l_@@_initial_open_bool }
4111             {
4112                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4113                 { \bool_set_true:N \l_@@_initial_open_bool }
4114             }
4115         }
4116     }
4117     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4118     {

```

```

4119         \int_compare:nNnT { #4 } = \c_one_int
4120         { \bool_set_true:N \l_@@_initial_open_bool }
4121     }
4122     {
4123         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4124         {
4125             \int_compare:nNnT { #4 } = { -1 }
4126             { \bool_set_true:N \l_@@_initial_open_bool }
4127         }
4128     }
4129 }
4130 \bool_if:NTF \l_@@_initial_open_bool
4131 {
4132     \int_add:Nn \l_@@_initial_i_int { #3 }
4133     \int_add:Nn \l_@@_initial_j_int { #4 }
4134     \bool_set_true:N \l_@@_stop_loop_bool
4135 }
4136 {
4137     \cs_if_exist:cTF
4138     {
4139         @@ _ dotted _
4140         \int_use:N \l_@@_initial_i_int -
4141         \int_use:N \l_@@_initial_j_int
4142     }
4143     {
4144         \int_add:Nn \l_@@_initial_i_int { #3 }
4145         \int_add:Nn \l_@@_initial_j_int { #4 }
4146         \bool_set_true:N \l_@@_initial_open_bool
4147         \bool_set_true:N \l_@@_stop_loop_bool
4148     }
4149     {
4150         \cs_if_exist:cTF
4151         {
4152             pgf @ sh @ ns @ \@@_env:
4153             - \int_use:N \l_@@_initial_i_int
4154             - \int_use:N \l_@@_initial_j_int
4155         }
4156         { \bool_set_true:N \l_@@_stop_loop_bool }
4157         {
4158             \cs_set:cpn
4159             {
4160                 @@ _ dotted _
4161                 \int_use:N \l_@@_initial_i_int -
4162                 \int_use:N \l_@@_initial_j_int
4163             }
4164             { }
4165         }
4166     }
4167 }
4168 }

```

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.

```

4169     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4170     {
4171         { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4172         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4173         { \int_use:N \l_@@_final_i_int }
4174         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4175         { } % for the name of the block
4176     }

```

```
4177 }
```

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.

```
4178 \cs_new_protected:Npn \@@_open_shorten:
4179 {
4180   \bool_if:NT \l_@@_initial_open_bool
4181   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4182   \bool_if:NT \l_@@_final_open_bool
4183   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4184 }
```

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).

```
4185 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4186 {
4187   \int_set:Nn \l_@@_row_min_int 1
4188   \int_set:Nn \l_@@_col_min_int 1
4189   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4190   \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`.

```
4191 \seq_map_inline:Nn \g_@@_submatrix_seq
4192 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4193 }
```

`#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.

```
4194 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4195 {
4196   \int_compare:nNnF { #3 } > { #1 }
4197   {
4198     \int_compare:nNnF { #1 } > { #5 }
4199     {
4200       \int_compare:nNnF { #4 } > { #2 }
4201       {
4202         \int_compare:nNnF { #2 } > { #6 }
4203         {
4204           \int_set:Nn \l_@@_row_min_int
4205           { \int_max:nn \l_@@_row_min_int { #3 } }
4206           \int_set:Nn \l_@@_col_min_int
4207           { \int_max:nn \l_@@_col_min_int { #4 } }
4208           \int_set:Nn \l_@@_row_max_int
4209           { \int_min:nn \l_@@_row_max_int { #5 } }
4210           \int_set:Nn \l_@@_col_max_int
4211           { \int_min:nn \l_@@_col_max_int { #6 } }
4212         }
4213       }
4214     }
4215   }
4216 }

4217 \cs_new_protected:Npn \@@_set_initial_coords:
4218 {
```

```

4219 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4220 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4221 }
4222 \cs_new_protected:Npn \@@_set_final_coords:
4223 {
4224 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4225 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4226 }
4227 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4228 {
4229 \pgfpointanchor
4230 {
4231 \@@_env:
4232 - \int_use:N \l_@@_initial_i_int
4233 - \int_use:N \l_@@_initial_j_int
4234 }
4235 { #1 }
4236 \@@_set_initial_coords:
4237 }
4238 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4239 {
4240 \pgfpointanchor
4241 {
4242 \@@_env:
4243 - \int_use:N \l_@@_final_i_int
4244 - \int_use:N \l_@@_final_j_int
4245 }
4246 { #1 }
4247 \@@_set_final_coords:
4248 }
4249 \cs_new_protected:Npn \@@_open_x_initial_dim:
4250 {
4251 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4252 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4253 {
4254 \cs_if_exist:cT
4255 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4256 {
4257 \pgfpointanchor
4258 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4259 { west }
4260 \dim_set:Nn \l_@@_x_initial_dim
4261 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4262 }
4263 }

```

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

```

4264 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4265 {
4266 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4267 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4268 \dim_add:Nn \l_@@_x_initial_dim \col@sep
4269 }
4270 }
4271 \cs_new_protected:Npn \@@_open_x_final_dim:
4272 {
4273 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4274 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4275 {
4276 \cs_if_exist:cT
4277 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4278 {
4279 \pgfpointanchor

```



```

4280         { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4281         { east }
4282         \dim_set:Nn \l_@@_x_final_dim
4283         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4284     }
4285 }

```

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

```

4286     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4287     {
4288         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4289         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4290         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4291     }
4292 }

```

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.

```

4293 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4294 {
4295     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4296     \cs_if_free:cT { @ _ dotted _ #1 - #2 }
4297     {
4298         \l_@@_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.

```

4299     \group_begin:
4300     \l_@@_open_shorten:
4301     \int_if_zero:nTF { #1 }
4302     { \color { nicematrix-first-row } }
4303     {

```

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.

```

4304         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4305         { \color { nicematrix-last-row } }
4306     }
4307     \keys_set:nn { NiceMatrix / xdots } { #3 }
4308     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4309     \l_@@_actually_draw_Ldots:
4310     \group_end:
4311 }
4312 }

```

The command `\l_@@_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`.

```

4313 \cs_new_protected:Npn \l_@@_actually_draw_Ldots:
4314 {
4315     \bool_if:NTF \l_@@_initial_open_bool
4316     {

```

```

4317 \@@_open_x_initial_dim:
4318 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4319 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4320 }
4321 { \@@_set_initial_coords_from_anchor:n { base~east } }
4322 \bool_if:NTF \l_@@_final_open_bool
4323 {
4324 \@@_open_x_final_dim:
4325 \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4326 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4327 }
4328 { \@@_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.

```

4329 \bool_lazy_all:nTF
4330 {
4331 \l_@@_initial_open_bool
4332 \l_@@_final_open_bool
4333 { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4334 }
4335 {
4336 \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4337 \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4338 }

```

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

```

4339 {
4340 \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4341 \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4342 }
4343 \@@_draw_line:
4344 }

```

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.

```

4345 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4346 {
4347 \@@_adjust_to_submatrix:nn { #1 } { #2 }
4348 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4349 {
4350 \@@_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.

```

4351 \group_begin:
4352 \@@_open_shorten:
4353 \int_if_zero:nTF { #1 }
4354 { \color { nicematrix-first-row } }
4355 {

```

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.

```

4356 \int_compare:nNnT { #1 } = \l_@@_last_row_int
4357 { \color { nicematrix-last-row } }
4358 }
4359 \keys_set:nn { NiceMatrix / xdots } { #3 }
4360 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4361 \@@_actually_draw_Cdots:
4362 \group_end:
4363 }
4364 }

```

The command `\l_@@_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.`

```

4365 \cs_new_protected:Npn \l_@@_actually_draw_Cdots:
4366 {
4367   \bool_if:NTF \l_@@_initial_open_bool
4368     { \@@_open_x_initial_dim: }
4369     { \@@_set_initial_coords_from_anchor:n { mid~east } }
4370   \bool_if:NTF \l_@@_final_open_bool
4371     { \@@_open_x_final_dim: }
4372     { \@@_set_final_coords_from_anchor:n { mid~west } }
4373   \bool_lazy_and:nnTF
4374     \l_@@_initial_open_bool
4375     \l_@@_final_open_bool
4376   {
4377     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4378     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4379     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4380     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4381     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4382   }
4383   {
4384     \bool_if:NT \l_@@_initial_open_bool
4385       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4386     \bool_if:NT \l_@@_final_open_bool
4387       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4388   }
4389   \@@_draw_line:
4390 }

4391 \cs_new_protected:Npn \l_@@_open_y_initial_dim:
4392 {
4393   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4394   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4395   {
4396     \cs_if_exist:cT
4397       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4398     {
4399       \pgfpointanchor
4400         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4401         { north }
4402       \dim_set:Nn \l_@@_y_initial_dim
4403         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4404     }
4405   }
4406   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4407   {
4408     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4409     \dim_set:Nn \l_@@_y_initial_dim
4410     {
4411       \fp_to_dim:n
4412       {
4413         \pgf@y
4414         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4415       }

```

```

4416     }
4417   }
4418 }
4419 \cs_new_protected:Npn \@@_open_y_final_dim:
4420 {
4421   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4422   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4423   {
4424     \cs_if_exist:cT
4425     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4426     {
4427       \pgfpointanchor
4428       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4429       { south }
4430       \dim_set:Nn \l_@@_y_final_dim
4431       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4432     }
4433   }
4434   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4435   {
4436     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4437     \dim_set:Nn \l_@@_y_final_dim
4438     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4439   }
4440 }

```

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.

```

4441 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4442 {
4443   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4444   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4445   {
4446     \@@_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.

```

4447   \group_begin:
4448   \@@_open_shorten:
4449   \int_if_zero:nTF { #2 }
4450   { \color { nicematrix-first-col } }
4451   {
4452     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4453     { \color { nicematrix-last-col } }
4454   }
4455   \keys_set:nn { NiceMatrix / xdots } { #3 }
4456   \tl_if_empty:oF \l_@@_xdots_color_tl
4457   { \color { \l_@@_xdots_color_tl } }
4458   \@@_actually_draw_Vdots:
4459   \group_end:
4460 }
4461 }

```

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`.

```
4462 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4463 {
```

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

```
4464 \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.

```
4465 {
4466   \@@_open_y_initial_dim:
4467   \@@_open_y_final_dim:
4468   \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4469 {
4470   \@@_qpoint:n { col - 1 }
4471   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4472   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4473   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4474   \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4475 }
4476 {
4477   \bool_lazy_and:nnTF
4478   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4479   { \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”.

```
4480 {
4481   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4482   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4483   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4484   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4485   \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4486 }
```

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

```
4487 {
4488   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4489   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4490   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4491   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4492 }
4493 }
4494 }
```

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.

```
4495 {
4496   \bool_set_false:N \l_tmpa_bool
4497   \bool_if:NF \l_@@_initial_open_bool
4498   {
4499     \bool_if:NF \l_@@_final_open_bool
4500     {
4501       \@@_set_initial_coords_from_anchor:n { south-west }
4502       \@@_set_final_coords_from_anchor:n { north-west }
4503       \bool_set:Nn \l_tmpa_bool
4504       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4505     }
4506   }
```

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

```
4507 \bool_if:NTF \l_@@_initial_open_bool
4508 {
4509   \@@_open_y_initial_dim:
```

```

4510     \@@_set_final_coords_from_anchor:n { north }
4511     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4512   }
4513   {
4514     \@@_set_initial_coords_from_anchor:n { south }
4515     \bool_if:NTF \l_@@_final_open_bool
4516     \@@_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.

```

4517   {
4518     \@@_set_final_coords_from_anchor:n { north }
4519     \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4520     {
4521       \dim_set:Nn \l_@@_x_initial_dim
4522       {
4523         \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4524         \l_@@_x_initial_dim \l_@@_x_final_dim
4525       }
4526     }
4527   }
4528 }
4529 }
4530 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4531 \@@_draw_line:
4532 }

```

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.

```

4533 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4534 {
4535   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4536   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4537   {
4538     \@@_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.

```

4539   \group_begin:
4540   \@@_open_shorten:
4541   \keys_set:nn { NiceMatrix / xdots } { #3 }
4542   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4543   \@@_actually_draw_Ddots:
4544   \group_end:
4545 }
4546 }

```

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`.

```

4547 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4548 {
4549   \bool_if:NTF \l_@@_initial_open_bool
4550   {
4551     \@@_open_y_initial_dim:
4552     \@@_open_x_initial_dim:
4553   }
4554   { \@@_set_initial_coords_from_anchor:n { south-east } }
4555   \bool_if:NTF \l_@@_final_open_bool
4556   {
4557     \@@_open_x_final_dim:
4558     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4559   }
4560   { \@@_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.

```

4561   \bool_if:NT \l_@@_parallelize_diags_bool
4562   {
4563     \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).

```

4564     \int_compare:nNnTF \g_@@_ddots_int = \c_one_int

```

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

```

4565     {
4566       \dim_gset:Nn \g_@@_delta_x_one_dim
4567       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4568       \dim_gset:Nn \g_@@_delta_y_one_dim
4569       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4570     }

```

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`.

```

4571     {
4572       \dim_set:Nn \l_@@_y_final_dim
4573       {
4574         \l_@@_y_initial_dim +
4575         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4576         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4577       }
4578     }
4579   }
4580   \@@_draw_line:
4581 }

```

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.

```

4582 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4583 {
4584   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4585   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4586   {
4587     \@@_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.

```

4588   \group_begin:
4589   \@@_open_shorten:
4590   \keys_set:nn { NiceMatrix / xdots } { #3 }
4591   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }

```

```

4592         \@@_actually_draw_Iddots:
4593     \group_end:
4594 }
4595 }

```

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.`

```

4596 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4597 {
4598     \bool_if:NTF \l_@@_initial_open_bool
4599     {
4600         \@@_open_y_initial_dim:
4601         \@@_open_x_initial_dim:
4602     }
4603     { \@@_set_initial_coords_from_anchor:n { south~west } }
4604     \bool_if:NTF \l_@@_final_open_bool
4605     {
4606         \@@_open_y_final_dim:
4607         \@@_open_x_final_dim:
4608     }
4609     { \@@_set_final_coords_from_anchor:n { north~east } }
4610     \bool_if:NT \l_@@_parallelize_diags_bool
4611     {
4612         \int_gincr:N \g_@@_iddots_int
4613         \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4614         {
4615             \dim_gset:Nn \g_@@_delta_x_two_dim
4616             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4617             \dim_gset:Nn \g_@@_delta_y_two_dim
4618             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4619         }
4620         {
4621             \dim_set:Nn \l_@@_y_final_dim
4622             {
4623                 \l_@@_y_initial_dim +
4624                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4625                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4626             }
4627         }
4628     }
4629     \@@_draw_line:
4630 }

```

18 The actual instructions for drawing the dotted lines with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

4631 \cs_new_protected:Npn \@@_draw_line:
4632 {
4633   \pgfrememberpicturepositiononpagetrue
4634   \pgf@relevantforpicturesizefalse
4635   \bool_lazy_or:nnTF
4636     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4637     \l_@@_dotted_bool
4638     \@@_draw_standard_dotted_line:
4639     \@@_draw_unstandard_dotted_line:
4640 }

```

We have to do a special construction with `\exp_args:No` to be able to put in the list of options in the correct place in the Tikz instruction.

```

4641 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4642 {
4643   \begin { scope }
4644     \@@_draw_unstandard_dotted_line:o
4645     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4646 }

```

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.

```

4647 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4648 {
4649   \@@_draw_unstandard_dotted_line:nooo
4650   { #1 }
4651   \l_@@_xdots_up_tl
4652   \l_@@_xdots_down_tl
4653   \l_@@_xdots_middle_tl
4654 }
4655 \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 continous line with a non-standard style.

```

4656 \hook_gput_code:nnn { begindocument } { . }
4657 {
4658   \IfPackageLoadedTF { tikz }
4659   {
4660     \tikzset
4661     {
4662       @@_node_above / .style = { sloped , above } ,
4663       @@_node_below / .style = { sloped , below } ,
4664       @@_node_middle / .style =
4665       {
4666         sloped ,
4667         inner~sep = \c_@@_innersep_middle_dim
4668       }
4669     }
4670   }
4671   { }
4672 }

```

```

4673 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4674 {

```

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.

```

4675 \dim_zero_new:N \l_@@_l_dim
4676 \dim_set:Nn \l_@@_l_dim
4677 {
4678   \fp_to_dim:n
4679   {
4680     sqrt
4681     (
4682       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4683       +
4684       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4685     )
4686   }
4687 }

```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the `aux` file to say that one more compilation should be done.

```

4688 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4689 {
4690   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4691   \@@_draw_unstandard_dotted_line_i:
4692 }

```

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

```

4693 \bool_if:NT \l_@@_xdots_h_labels_bool
4694 {
4695   \tikzset
4696   {
4697     @@_node_above / .style = { auto = left } ,
4698     @@_node_below / .style = { auto = right } ,
4699     @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4700   }
4701 }
4702 \tl_if_empty:nF { #4 }
4703 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4704 \draw
4705 [ #1 ]
4706 ( \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).

```

4707 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4708 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4709 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4710 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4711 \end { scope }
4712 }
4713 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4714 {
4715   \dim_set:Nn \l_tmpa_dim
4716   {
4717     \l_@@_x_initial_dim
4718     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4719     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4720 }
4721 \dim_set:Nn \l_tmpb_dim
4722 {
4723   \l_@@_y_initial_dim
4724   + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4725   * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4726 }
4727 \dim_set:Nn \l_@@_tmpc_dim
4728 {
4729   \l_@@_x_final_dim
4730   - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4731   * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4732 }
4733 \dim_set:Nn \l_@@_tmpd_dim
4734 {
4735   \l_@@_y_final_dim
4736   - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4737   * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4738 }
4739 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4740 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4741 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4742 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4743 }
4744 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

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

```

4745 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4746 {
4747   \group_begin:

```

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

```

4748   \dim_zero_new:N \l_@@_l_dim
4749   \dim_set:Nn \l_@@_l_dim
4750   {
4751     \fp_to_dim:n
4752     {
4753       sqrt
4754       (
4755         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4756         +
4757         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4758       )
4759     }
4760   }

```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the aux file to say that one more compilation should be done.

```

4761   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4762   {
4763     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4764     \@@_draw_standard_dotted_line_i:
4765   }
4766 \group_end:
4767 \bool_lazy_all:nF
4768 {
4769   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4770   { \tl_if_empty_p:N \l_@@_xdots_down_tl }

```

```

4771      { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4772    }
4773    \l_@@_labels_standard_dotted_line:
4774  }
4775  \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4776  \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4777  {

```

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

```

4778    \int_set:Nn \l_tmpa_int
4779    {
4780      \dim_ratio:nn
4781      {
4782        \l_@@_l_dim
4783        - \l_@@_xdots_shorten_start_dim
4784        - \l_@@_xdots_shorten_end_dim
4785      }
4786      \l_@@_xdots_inter_dim
4787    }

```

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

```

4788    \dim_set:Nn \l_tmpa_dim
4789    {
4790      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4791      \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4792    }
4793    \dim_set:Nn \l_tmpb_dim
4794    {
4795      ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4796      \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4797    }

```

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.

```

4798    \dim_gadd:Nn \l_@@_x_initial_dim
4799    {
4800      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4801      \dim_ratio:nn
4802      {
4803        \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4804        + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4805      }
4806      { 2 \l_@@_l_dim }
4807    }
4808    \dim_gadd:Nn \l_@@_y_initial_dim
4809    {
4810      ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4811      \dim_ratio:nn
4812      {
4813        \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4814        + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4815      }
4816      { 2 \l_@@_l_dim }
4817    }
4818    \pgf@relevantforpicturesizefalse
4819    \int_step_inline:nnn \c_zero_int \l_tmpa_int
4820    {
4821      \pgfpathcircle
4822      { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4823      { \l_@@_xdots_radius_dim }
4824      \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4825      \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim

```

```

4826     }
4827     \pgfusepathqfill
4828 }

4829 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4830 {
4831     \pgfscope
4832     \pgftransformshift
4833     {
4834         \pgfpointlineattime { 0.5 }
4835         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4836         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4837     }
4838     \fp_set:Nn \l_tmpa_fp
4839     {
4840         atand
4841         (
4842             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4843             \l_@@_x_final_dim - \l_@@_x_initial_dim
4844         )
4845     }
4846     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4847     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4848     \tl_if_empty:NF \l_@@_xdots_middle_tl
4849     {
4850         \begin { pgfscope }
4851         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4852         \pgfnode
4853         { rectangle }
4854         { center }
4855         {
4856             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4857             {
4858                 \c_math_toggle_token
4859                 \scriptstyle \l_@@_xdots_middle_tl
4860                 \c_math_toggle_token
4861             }
4862         }
4863         { }
4864         {
4865             \pgfsetfillcolor { white }
4866             \pgfusepath { fill }
4867         }
4868         \end { pgfscope }
4869     }
4870     \tl_if_empty:NF \l_@@_xdots_up_tl
4871     {
4872         \pgfnode
4873         { rectangle }
4874         { south }
4875         {
4876             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4877             {
4878                 \c_math_toggle_token
4879                 \scriptstyle \l_@@_xdots_up_tl
4880                 \c_math_toggle_token
4881             }
4882         }
4883         { }
4884         { \pgfusepath { } }
4885     }
4886     \tl_if_empty:NF \l_@@_xdots_down_tl
4887     {

```

```

4888 \pgfnode
4889 { rectangle }
4890 { north }
4891 {
4892   \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4893   {
4894     \c_math_toggle_token
4895     \scriptstyle \l_@@_xdots_down_tl
4896     \c_math_toggle_token
4897   }
4898 }
4899 { }
4900 { \pgfusepath { } }
4901 }
4902 \endpgfscope
4903 }

```

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.

```

4904 \hook_gput_code:nnn { begindocument } { . }
4905 {
4906   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4907   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4908   \cs_new_protected:Npn \@@_Ldots
4909     { \@@_collect_options:n { \@@_Ldots_i } }
4910   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4911     {
4912       \int_if_zero:nTF \c@jCol
4913       { \@@_error:nn { in~first~col } \Ldots }
4914       {
4915         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4916         { \@@_error:nn { in~last~col } \Ldots }
4917         {
4918           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4919           { #1 , down = #2 , up = #3 , middle = #4 }
4920         }
4921       }
4922       \bool_if:NF \l_@@_nullify_dots_bool
4923       { \phantom { \ensuremath { \@@_old_ldots } } }
4924       \bool_gset_true:N \g_@@_empty_cell_bool
4925     }

4926   \cs_new_protected:Npn \@@_Cdots
4927     { \@@_collect_options:n { \@@_Cdots_i } }
4928   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4929     {
4930       \int_if_zero:nTF \c@jCol
4931       { \@@_error:nn { in~first~col } \Cdots }
4932       {

```

```

4933     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4934     { \@@_error:nn { in~last~col } \Cdots }
4935     {
4936         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4937         { #1 , down = #2 , up = #3 , middle = #4 }
4938     }
4939 }
4940 \bool_if:NF \l_@@_nullify_dots_bool
4941 { \phantom { \ensuremath { \@@_old_cdots } } }
4942 \bool_gset_true:N \g_@@_empty_cell_bool
4943 }

4944 \cs_new_protected:Npn \@@_Vdots
4945 { \@@_collect_options:n { \@@_Vdots_i } }
4946 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4947 {
4948     \int_if_zero:nTF \c@iRow
4949     { \@@_error:nn { in~first~row } \Vdots }
4950     {
4951         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4952         { \@@_error:nn { in~last~row } \Vdots }
4953         {
4954             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4955             { #1 , down = #2 , up = #3 , middle = #4 }
4956         }
4957     }
4958     \bool_if:NF \l_@@_nullify_dots_bool
4959     { \phantom { \ensuremath { \@@_old_vdots } } }
4960     \bool_gset_true:N \g_@@_empty_cell_bool
4961 }

4962 \cs_new_protected:Npn \@@_Ddots
4963 { \@@_collect_options:n { \@@_Ddots_i } }
4964 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4965 {
4966     \int_case:nnF \c@iRow
4967     {
4968         0 { \@@_error:nn { in~first~row } \Ddots }
4969         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4970     }
4971     {
4972         \int_case:nnF \c@jCol
4973         {
4974             0 { \@@_error:nn { in~first~col } \Ddots }
4975             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4976         }
4977         {
4978             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4979             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4980             { #1 , down = #2 , up = #3 , middle = #4 }
4981         }
4982     }
4983 }
4984 \bool_if:NF \l_@@_nullify_dots_bool
4985 { \phantom { \ensuremath { \@@_old_ddots } } }
4986 \bool_gset_true:N \g_@@_empty_cell_bool
4987 }

4988 \cs_new_protected:Npn \@@_Iddots
4989 { \@@_collect_options:n { \@@_Iddots_i } }
4990 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl

```

```

4991 {
4992   \int_case:nnF \c@iRow
4993   {
4994     0 { \@@_error:nn { in-first~row } \Iddots }
4995     \l_@@_last_row_int { \@@_error:nn { in-last~row } \Iddots }
4996   }
4997   {
4998     \int_case:nnF \c@jCol
4999     {
5000       0 { \@@_error:nn { in-first~col } \Iddots }
5001       \l_@@_last_col_int { \@@_error:nn { in-last~col } \Iddots }
5002     }
5003     {
5004       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5005       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5006       { #1 , down = #2 , up = #3 , middle = #4 }
5007     }
5008   }
5009   \bool_if:NF \l_@@_nullify_dots_bool
5010   { \phantom { \ensuremath { \@@_old_iddots } } }
5011   \bool_gset_true:N \g_@@_empty_cell_bool
5012 }
5013 }

```

End of the `\AddToHook`.

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

```

5014 \keys_define:nn { NiceMatrix / Ddots }
5015 {
5016   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5017   draw-first .default:n = true ,
5018   draw-first .value_forbidden:n = true
5019 }

```

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

```

5020 \cs_new_protected:Npn \@@_Hspace:
5021 {
5022   \bool_gset_true:N \g_@@_empty_cell_bool
5023   \hspace
5024 }

```

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`.

```

5025 \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`.

```

5026 \cs_new:Npn \@@_Hdotsfor:
5027 {
5028   \bool_lazy_and:nnTF
5029   { \int_if_zero_p:n \c@jCol }
5030   { \int_if_zero_p:n \l_@@_first_col_int }
5031   {
5032     \bool_if:NTF \g_@@_after_col_zero_bool
5033     {
5034       \multicolumn { 1 } { c } { }
5035       \@@_Hdotsfor_i
5036     }
5037     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5038   }

```



```

5039     {
5040     \multicolumn { 1 } { c } { }
5041     \@@_Hdotsfor_i
5042     }
5043 }

```

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_i`).

```

5044 \hook_gput_code:nnn { begindocument } { . }
5045 {
5046   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5047   \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.

```

5048   \cs_new_protected:Npn \@@_Hdotsfor_i
5049   { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5050   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5051   {
5052     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5053     {
5054       \@@_Hdotsfor:nnnn
5055       { \int_use:N \c@iRow }
5056       { \int_use:N \c@jCol }
5057       { #2 }
5058       {
5059         #1 , #3 ,
5060         down = \exp_not:n { #4 } ,
5061         up = \exp_not:n { #5 } ,
5062         middle = \exp_not:n { #6 }
5063       }
5064     }
5065     \prg_replicate:nn { #2 - 1 }
5066     {
5067       &
5068       \multicolumn { 1 } { c } { }
5069       \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5070     }
5071   }
5072 }

```

```

5073 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5074 {
5075   \bool_set_false:N \l_@@_initial_open_bool
5076   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5077   \int_set:Nn \l_@@_initial_i_int { #1 }
5078   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5079   \int_compare:nNnTF { #2 } = \c_one_int
5080   {
5081     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5082     \bool_set_true:N \l_@@_initial_open_bool
5083   }
5084   {
5085     \cs_if_exist:cTF
5086     {
5087       pgf @ sh @ ns @ \@@_env:
5088       - \int_use:N \l_@@_initial_i_int
5089       - \int_eval:n { #2 - 1 }
5090     }
5091     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }

```

```

5092     {
5093         \int_set:Nn \l_@@_initial_j_int { #2 }
5094         \bool_set_true:N \l_@@_initial_open_bool
5095     }
5096 }
5097 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5098 {
5099     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5100     \bool_set_true:N \l_@@_final_open_bool
5101 }
5102 {
5103     \cs_if_exist:cTF
5104     {
5105         pgf @ sh @ ns @ \@@_env:
5106         - \int_use:N \l_@@_final_i_int
5107         - \int_eval:n { #2 + #3 }
5108     }
5109     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5110     {
5111         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5112         \bool_set_true:N \l_@@_final_open_bool
5113     }
5114 }
5115 \group_begin:
5116 \@@_open_shorten:
5117 \int_if_zero:nTF { #1 }
5118 { \color { nicematrix-first-row } }
5119 {
5120     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5121     { \color { nicematrix-last-row } }
5122 }
5123
5124 \keys_set:nn { NiceMatrix / xdots } { #4 }
5125 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5126 \@@_actually_draw_Ldots:
5127 \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).

```

5128     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5129     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5130 }

5131 \hook_gput_code:nnn { begindocument } { . }
5132 {
5133     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5134     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5135     \cs_new_protected:Npn \@@_Vdotsfor:
5136     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5137     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5138     {
5139         \bool_gset_true:N \g_@@_empty_cell_bool
5140         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5141         {
5142             \@@_Vdotsfor:nnnn
5143             { \int_use:N \c@iRow }
5144             { \int_use:N \c@jCol }
5145             { #2 }
5146             {
5147                 #1 , #3 ,
5148                 down = \exp_not:n { #4 } ,

```

```

5149         up = \exp_not:n { #5 } ,
5150         middle = \exp_not:n { #6 }
5151     }
5152 }
5153 }
5154 }

```

```

5155 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5156 {
5157     \bool_set_false:N \l_@@_initial_open_bool
5158     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5159     \int_set:Nn \l_@@_initial_j_int { #2 }
5160     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5161     \int_compare:nNnTF { #1 } = \c_one_int
5162     {
5163         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5164         \bool_set_true:N \l_@@_initial_open_bool
5165     }
5166     {
5167         \cs_if_exist:cTF
5168         {
5169             pgf @ sh @ ns @ \@@_env:
5170             - \int_eval:n { #1 - 1 }
5171             - \int_use:N \l_@@_initial_j_int
5172         }
5173         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5174         {
5175             \int_set:Nn \l_@@_initial_i_int { #1 }
5176             \bool_set_true:N \l_@@_initial_open_bool
5177         }
5178     }
5179     \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5180     {
5181         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5182         \bool_set_true:N \l_@@_final_open_bool
5183     }
5184     {
5185         \cs_if_exist:cTF
5186         {
5187             pgf @ sh @ ns @ \@@_env:
5188             - \int_eval:n { #1 + #3 }
5189             - \int_use:N \l_@@_final_j_int
5190         }
5191         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5192         {
5193             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5194             \bool_set_true:N \l_@@_final_open_bool
5195         }
5196     }
5197     \group_begin:
5198     \@@_open_shorten:
5199     \int_if_zero:nTF { #2 }
5200     { \color { nicematrix-first-col } }
5201     {
5202         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5203         { \color { nicematrix-last-col } }
5204     }
5205     \keys_set:nn { NiceMatrix / xdots } { #4 }
5206     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }

```

```

5207 \@@_actually_draw_Vdots:
5208 \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).

```

5209 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5210 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5211 }

```

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

```

5212 \NewDocumentCommand \@@_rotate: { 0 { } }
5213 {
5214   \peek_remove_spaces:n
5215   {
5216     \bool_gset_true:N \g_@@_rotate_bool
5217     \keys_set:nn { NiceMatrix / rotate } { #1 }
5218   }
5219 }

5220 \keys_define:nn { NiceMatrix / rotate }
5221 {
5222   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5223   c .value_forbidden:n = true ,
5224   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5225 }

```

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

```

5226 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5227 {
5228   \tl_if_empty:nTF { #2 }
5229   { #1 }
5230   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5231 }
5232 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5233 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

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

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).

```

5234 \hook_gput_code:nnn { begindocument } { . }
5235 {
5236   \cs_set_nopar:Npn \l_@@_argspec_tl
5237     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5238   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5239   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5240     {
5241       \group_begin:
5242       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5243       \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5244       \use:e
5245       {
5246         \@@_line_i:nn
5247         { \@@_double_int_eval:n #2 - \q_stop }
5248         { \@@_double_int_eval:n #3 - \q_stop }
5249       }
5250       \group_end:
5251     }
5252 }
5253 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5254 {
5255   \bool_set_false:N \l_@@_initial_open_bool
5256   \bool_set_false:N \l_@@_final_open_bool
5257   \bool_lazy_or:nnTF
5258     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5259     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5260     { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

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

```

5261   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5262 }
5263 \hook_gput_code:nnn { begindocument } { . }
5264 {
5265   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5266   {

```

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:`.

```

5267   \c_@@_pgfortikzpicture_tl
5268   \@@_draw_line_iii:nn { #1 } { #2 }
5269   \c_@@_endpgfortikzpicture_tl
5270 }
5271 }

```

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

```

5272 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5273 {
5274   \pgfrememberpicturepositiononpagetrue
5275   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5276   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5277   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5278   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5279   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5280   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5281   \@@_draw_line:
5282 }

```

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

21 The command `\RowStyle`

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

```
\@@_if_row_less_than:nn { number } { instructions }
```

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

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

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

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

```
5283 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5284 { \int_compare:nNtT { \c@iRow } < { #1 } { #2 } }
```

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

```
5285 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5286 {
5287   \tl_gput_right:Nx \g_@@_row_style_tl
5288   {
```

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

```
5289   \exp_not:N
5290   \@@_if_row_less_than:nn
5291   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
```

The `\scan_stop:` is mandatory (for ex. for the case where `\rotate` is used in the argument of `\RowStyle`).

```
5292   { \exp_not:n { #1 } \scan_stop: }
5293 }
5294 }
5295 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5296 \keys_define:nn { NiceMatrix / RowStyle }
5297 {
5298   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5299   cell-space-top-limit .value_required:n = true ,
5300   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5301   cell-space-bottom-limit .value_required:n = true ,
5302   cell-space-limits .meta:n =
5303   {
5304     cell-space-top-limit = #1 ,
5305     cell-space-bottom-limit = #1 ,
5306   } ,
5307   color .tl_set:N = \l_@@_color_tl ,
5308   color .value_required:n = true ,
5309   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5310   bold .default:n = true ,
5311   nb-rows .code:n =
5312   { \str_if_eq:nnTF { #1 } { * }
5313     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5314     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5315   nb-rows .value_required:n = true ,
5316   rowcolor .tl_set:N = \l_tmpa_tl ,
5317   rowcolor .value_required:n = true ,
5318   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5319 }
```

```

5320 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5321 {
5322   \group_begin:
5323   \tl_clear:N \l_tmpa_tl
5324   \tl_clear:N \l_@@_color_tl
5325   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5326   \dim_zero:N \l_tmpa_dim
5327   \dim_zero:N \l_tmpb_dim
5328   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

5329   \tl_if_empty:NF \l_tmpa_tl
5330   {

```

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

```

5331     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5332     {

```

The command \@@_exp_color_arg:No is *fully expandable*.

```

5333         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5334         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5335         { \int_use:N \c@iRow - * }
5336     }

```

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

```

5337     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5338     {
5339       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5340       {
5341         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5342         {
5343           \int_eval:n { \c@iRow + 1 }
5344           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5345         }
5346       }
5347     }
5348   }
5349   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5350   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5351   {
5352     \exp_args:Nx \@@_put_in_row_style:n
5353     {
5354       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5355       {

```

It's not possible to change the following code by using \dim_set_eq:NN (because of expansion).

```

5356         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5357         { \dim_use:N \l_tmpa_dim }
5358       }
5359     }
5360   }

```

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

```

5361   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5362   {
5363     \exp_args:Nx \@@_put_in_row_style:n
5364     {
5365       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5366       {
5367         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5368         { \dim_use:N \l_tmpb_dim }
5369       }
5370     }
5371   }

```

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

```

5372   \tl_if_empty:NF \l_@@_color_tl
5373   {
5374     \@@_put_in_row_style:e
5375     {
5376       \mode_leave_vertical:
5377       \@@_color:n { \l_@@_color_tl }
5378     }
5379   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5380   \bool_if:NT \l_@@_bold_row_style_bool
5381   {
5382     \@@_put_in_row_style:n
5383     {
5384       \exp_not:n
5385       {
5386         \if_mode_math:
5387           \c_math_toggle_token
5388           \bfseries \boldmath
5389           \c_math_toggle_token
5390         \else:
5391           \bfseries \boldmath
5392         \fi:
5393       }
5394     }
5395   }
5396   \group_end:
5397   \g_@@_row_style_tl
5398   \ignorespaces
5399 }

```

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).

```

5400 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5401 {

```


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.

```
5402 \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`.

```
5403 \str_if_in:nnF { #1 } { !! }
5404 {
5405   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5406   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5407 }
5408 \int_if_zero:nTF \l_tmpa_int
```

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

```
5409 {
5410   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5411   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5412 }
```

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

```
5413 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5414 }
5415 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5416 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5417 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5418 {
5419   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5420   {
```

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

```
5421   \group_begin:
5422   \pgfsetcornersarced
5423   {
5424     \pgfpoint
5425     { \l_@@_tab_rounded_corners_dim }
5426     { \l_@@_tab_rounded_corners_dim }
5427   }
```

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.

```
5428   \bool_if:NTF \l_@@_hvlines_bool
5429   {
5430     \pgfpathrectanglecorners
5431     {
5432       \pgfpointadd
5433       { \@@_qpoint:n { row-1 } }
5434       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5435     }
5436     {
5437       \pgfpointadd
5438       {
5439         \@@_qpoint:n
5440         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5441       }
5442       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5443     }
5444   }
```

```

5445     {
5446         \pgfpathrectanglecorners
5447         { \@@_qpoint:n { row-1 } }
5448         {
5449             \pgfpointadd
5450             {
5451                 \@@_qpoint:n
5452                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5453             }
5454             { \pgfpoint \c_zero_dim \arrayrulewidth }
5455         }
5456     }
5457     \pgfusepath { clip }
5458     \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5459     }
5460 }

```

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).

```

5461 \cs_new_protected:Npn \@@_actually_color:
5462 {
5463     \pgfpicture
5464     \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.

```

5465     \@@_clip_with_rounded_corners:
5466     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5467     {
5468         \int_compare:nNnTF { ##1 } = \c_one_int
5469         {
5470             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5471             \use:c { g_@@_color _ 1 _tl }
5472             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5473         }
5474         {
5475             \begin { pgfscope }
5476                 \@@_color_opacity ##2
5477                 \use:c { g_@@_color _ ##1 _tl }
5478                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5479                 \pgfusepath { fill }
5480             \end { pgfscope }
5481         }
5482     }
5483     \endpgfpicture
5484 }

```

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

```

5485 \cs_new_protected:Npn \@@_color_opacity
5486 {
5487     \peek_meaning:NTF [
5488         { \@@_color_opacity:w }
5489         { \@@_color_opacity:w [ ] }
5490     }

```

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.

```

5491 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5492 {

```

```

5493 \tl_clear:N \l_tmpa_tl
5494 \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.
5495 \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5496 \tl_if_empty:NTF \l_tmpb_tl
5497 { \@declaredcolor }
5498 { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5499 }

```

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

```

5500 \keys_define:nn { nicematrix / color-opacity }
5501 {
5502   opacity .tl_set:N          = \l_tmpa_tl ,
5503   opacity .value_required:n = true
5504 }

5505 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5506 {
5507   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5508   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5509   \@@_cartesian_path:
5510 }

```

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

```

5511 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5512 {
5513   \tl_if_blank:nF { #2 }
5514   {
5515     \@@_add_to_colors_seq:en
5516     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5517     { \@@_cartesian_color:nn { #3 } { - } }
5518   }
5519 }

```

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

```

5520 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5521 {
5522   \tl_if_blank:nF { #2 }
5523   {
5524     \@@_add_to_colors_seq:en
5525     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5526     { \@@_cartesian_color:nn { - } { #3 } }
5527   }
5528 }

```

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

```

5529 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5530 {
5531   \tl_if_blank:nF { #2 }
5532   {
5533     \@@_add_to_colors_seq:en
5534     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5535     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5536   }
5537 }

```

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

```

5538 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5539 {
5540   \tl_if_blank:nF { #2 }
5541   {
5542     \@@_add_to_colors_seq:en
5543     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5544     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5545   }
5546 }

```

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

```

5547 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5548 {
5549   \@@_cut_on_hyphen:w #1 \q_stop
5550   \tl_clear_new:N \l_@@_tmpc_tl
5551   \tl_clear_new:N \l_@@_tmpd_tl
5552   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5553   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5554   \@@_cut_on_hyphen:w #2 \q_stop
5555   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5556   \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`.

```

5557   \@@_cartesian_path:n { #3 }
5558 }

```

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

```

5559 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5560 {
5561   \clist_map_inline:nn { #3 }
5562   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5563 }

```

```

5564 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5565 {
5566   \int_step_inline:nn \c@iRow
5567   {
5568     \int_step_inline:nn \c@jCol
5569     {
5570       \int_if_even:nTF { ####1 + ##1 }
5571       { \@@_cellcolor [ #1 ] { #2 } }
5572       { \@@_cellcolor [ #1 ] { #3 } }
5573       { ##1 - ####1 }
5574     }
5575   }
5576 }

```

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”.

```

5577 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5578 {
5579   \@@_rectanglecolor [ #1 ] { #2 }
5580   { 1 - 1 }
5581   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5582 }

```

```

5583 \keys_define:nn { NiceMatrix / rowcolors }
5584 {
5585   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5586   respect-blocks .default:n = true ,
5587   cols .tl_set:N = \l_@@_cols_tl ,
5588   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5589   restart .default:n = true ,
5590   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5591 }

```

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*.

```

5592 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5593 {

```

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

```

5594   \group_begin:
5595   \seq_clear_new:N \l_@@_colors_seq
5596   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5597   \tl_clear_new:N \l_@@_cols_tl
5598   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5599   \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).

```

5600   \int_zero_new:N \l_@@_color_int
5601   \int_set_eq:NN \l_@@_color_int \c_one_int
5602   \bool_if:NT \l_@@_respect_blocks_bool
5603   {

```

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`).

```

5604       \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5605       \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5606       { \@@_not_in_exterior_p:nnnnn ##1 }
5607   }
5608   \pgfpicture
5609   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5610   \clist_map_inline:nn { #2 }
5611   {
5612     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5613     \tl_if_in:NnTF \l_tmpa_tl { - }
5614     { \@@_cut_on_hyphen:w ##1 \q_stop }
5615     { \tl_set:No \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.

```

5616     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5617     \int_set:Nn \l_@@_color_int
5618     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5619     \int_zero_new:N \l_@@_tmpc_int
5620     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5621     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5622     {

```

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

```

5623       \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).

```

5624     \bool_if:NT \l_@@_respect_blocks_bool
5625     {
5626         \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5627         { \@@_intersect_our_row_p:nnnnn ###1 }
5628         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

5629     }
5630     \tl_set:No \l_@@_rows_tl
5631     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5632     \tl_clear_new:N \l_@@_color_tl
5633     \tl_set:Nx \l_@@_color_tl
5634     {
5635         \@@_color_index:n
5636         {
5637             \int_mod:nn
5638             { \l_@@_color_int - 1 }
5639             { \seq_count:N \l_@@_colors_seq }
5640             + 1
5641         }
5642     }
5643     \tl_if_empty:NF \l_@@_color_tl
5644     {
5645         \@@_add_to_colors_seq:ee
5646         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5647         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5648     }
5649     \int_incr:N \l_@@_color_int
5650     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5651 }
5652 }
5653 \endpgfpicture
5654 \group_end:
5655 }

```

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.

```

5656 \cs_new:Npn \@@_color_index:n #1
5657 {
5658     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5659     { \@@_color_index:n { #1 - 1 } }
5660     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5661 }

```

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

```

5662 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5663 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

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

```

5664 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5665 {
5666     \int_compare:nNnT { #3 } > \l_tmpb_int
5667     { \int_set:Nn \l_tmpb_int { #3 } }
5668 }

```

```

5669 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5670 {
5671   \int_if_zero:nTF { #4 }
5672   \prg_return_false:
5673   {
5674     \int_compare:nNnTF { #2 } > \c@jCol
5675     \prg_return_false:
5676     \prg_return_true:
5677   }
5678 }

```

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

```

5679 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5680 {
5681   \int_compare:nNnTF { #1 } > \l_tmpa_int
5682   \prg_return_false:
5683   {
5684     \int_compare:nNnTF \l_tmpa_int > { #3 }
5685     \prg_return_false:
5686     \prg_return_true:
5687   }
5688 }

```

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).

```

5689 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5690 {
5691   \dim_compare:nNnTF { #1 } = \c_zero_dim
5692   {
5693     \bool_if:NTF
5694     \l_@@_nocolor_used_bool
5695     \@@_cartesian_path_normal_ii:
5696     {
5697       \seq_if_empty:NTF \l_@@_corners_cells_seq
5698       { \@@_cartesian_path_normal_i:n { #1 } }
5699       \@@_cartesian_path_normal_ii:
5700     }
5701   }
5702   { \@@_cartesian_path_normal_i:n { #1 } }
5703 }

```

First, the situation where is a rectangular zone of cells will be colored as a whole (in the instructions of the resulting PDF). The argument is the radius of the corners.

```

5704 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5705 {
5706   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5707 \clist_map_inline:Nn \l_@@_cols_tl
5708 {
5709   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5710   \tl_if_in:NnTF \l_tmpa_tl { - }
5711   { \@@_cut_on_hyphen:w ##1 \q_stop }
5712   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5713   \tl_if_empty:NTF \l_tmpa_tl
5714   { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5715   {

```

```

5716         \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5717         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5718     }
5719     \tl_if_empty:NTF \l_tmpb_tl
5720     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5721     {
5722         \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5723         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5724     }
5725     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5726     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5727     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5728     \@@_qpoint:n { col - \l_tmpa_tl }
5729     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5730     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5731     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5732     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5733     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5734     \clist_map_inline:Nn \l_@@_rows_tl
5735     {
5736         \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5737         \tl_if_in:NnTF \l_tmpa_tl { - }
5738         { \@@_cut_on_hyphen:w #####1 \q_stop }
5739         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5740         \tl_if_empty:NTF \l_tmpa_tl
5741         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5742         {
5743             \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5744             { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5745         }
5746         \tl_if_empty:NTF \l_tmpb_tl
5747         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5748         {
5749             \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5750             { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5751         }
5752         \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5753         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_row_total_int } }

```

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

```

5754         \cs_if_exist:cF
5755         { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5756         {
5757             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5758             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5759             \@@_qpoint:n { row - \l_tmpa_tl }
5760             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5761             \pgfpathrectanglecorners
5762             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5763             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5764         }
5765     }
5766 }
5767 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5768 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5769 {
5770     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5771     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```


We begin the loop over the columns.

```

5772 \clist_map_inline:Nn \l_@@_cols_tl
5773 {
5774   \@@_qpoint:n { col - ##1 }
5775   \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5776     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5777     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5778   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5779   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5780 \clist_map_inline:Nn \l_@@_rows_tl
5781 {
5782   \seq_if_in:NnF \l_@@_corners_cells_seq
5783   { #####1 - ##1 }
5784   {
5785     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5786     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5787     \@@_qpoint:n { row - #####1 }
5788     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5789     \cs_if_exist:cF { @@ _ #####1 _ ##1 _ nocolor }
5790     {
5791       \pgfpathrectanglecorners
5792       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5793       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5794     }
5795   }
5796 }
5797 }
5798 }

```

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`).

```

5799 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n \c_zero_dim }

```

Despite its name, the following command does not create a PGF path. It declares as colored by the “empty color” all the cells in what would be the path. Hence, the other coloring instructions of `nicematrix` won’t put color in those cells. the

```

5800 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5801 {
5802   \bool_set_true:N \l_@@_nocolor_used_bool
5803   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5804   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5805 \clist_map_inline:Nn \l_@@_rows_tl
5806 {
5807   \clist_map_inline:Nn \l_@@_cols_tl
5808   { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5809 }
5810 }

```

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 `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

5811 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5812 {
5813   \clist_set_eq:NN \l_tmpa_clist #1
5814   \clist_clear:N #1
5815   \clist_map_inline:Nn \l_tmpa_clist
5816   {
5817     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }

```

```

5818 \tl_if_in:NnTF \l_tmpa_tl { - }
5819 { \@@_cut_on_hyphen:w ##1 \q_stop }
5820 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5821 \bool_lazy_or:nnT
5822 { \tl_if_blank_p:o \l_tmpa_tl }
5823 { \str_if_eq_p:on \l_tmpa_tl { * } }
5824 { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5825 \bool_lazy_or:nnT
5826 { \tl_if_blank_p:o \l_tmpb_tl }
5827 { \str_if_eq_p:on \l_tmpb_tl { * } }
5828 { \tl_set:N \l_tmpb_tl { \int_use:N #2 } }
5829 \int_compare:nNnT \l_tmpb_tl > #2
5830 { \tl_set:N \l_tmpb_tl { \int_use:N #2 } }
5831 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5832 { \clist_put_right:Nn #1 { ####1 } }
5833 }
5834 }

```

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

```

5835 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5836 {
5837   \@@_test_color_inside:
5838   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5839   {

```

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`).

```

5840     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5841     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5842   }
5843   \ignorespaces
5844 }

```

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

```

5845 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5846 {
5847   \@@_test_color_inside:
5848   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5849   {
5850     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5851     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5852     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5853   }
5854   \ignorespaces
5855 }

```

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).

```

5856 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5857 { \@@_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.

```

5858 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5859 {
5860   \@@_test_color_inside:
5861   \peek_remove_spaces:n
5862   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5863 }

```

```

5864 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5865 {

```

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`.

```

5866 \seq_gclear:N \g_tmpa_seq
5867 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5868 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5869 \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`.

```

5870 \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5871 {
5872   { \int_use:N \c@iRow }
5873   { \exp_not:n { #1 } }
5874   { \exp_not:n { #2 } }
5875   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5876 }
5877 }

```

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`).

```

5878 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5879 {
5880   \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).

```

5881   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5882   {
5883     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5884     {
5885       \@@_rowlistcolors
5886       [ \exp_not:n { #2 } ]
5887       { #1 - \int_eval:n { \c@iRow - 1 } }
5888       { \exp_not:n { #3 } }
5889       [ \exp_not:n { #4 } ]
5890     }
5891   }
5892 }

```

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.

```

5893 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5894 {
5895   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5896   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5897   \seq_gclear:N \g_@@_rowlistcolors_seq
5898 }

```

```

5899 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5900 {
5901   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5902   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5903 }

```

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.

```

5904 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5905 {

```

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).

```

5906   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5907   {

```

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).

```

5908     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5909     {
5910       \exp_not:N \columncolor [ #1 ]
5911       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5912     }
5913   }
5914 }

```

```

5915 \hook_gput_code:nnn { begindocument } { . }
5916 {
5917   \IfPackageLoadedTF { colortbl }
5918   {
5919     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5920     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5921     \cs_new_protected:Npn \@@_revert_colortbl:
5922     {
5923       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5924       {
5925         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5926         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5927       }
5928     }
5929   }
5930   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5931 }

```

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`.

```
5932 \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`.

```
5933 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5934 {
5935   \int_if_zero:nTF \l_@@_first_col_int
5936   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5937   {
5938     \int_if_zero:nTF \c@jCol
5939     {
5940       \int_compare:nNnF \c@iRow = { -1 }
5941       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5942     }
5943     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5944   }
5945 }
```

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.

```
5946 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5947 {
5948   \int_if_zero:nF \c@iRow
5949   {
5950     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5951     {
5952       \int_compare:nNnT \c@jCol > \c_zero_int
5953       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5954     }
5955   }
5956 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
5957 \keys_define:nn { NiceMatrix / Rules }
5958 {
5959   position .int_set:N = \l_@@_position_int ,
5960   position .value_required:n = true ,
5961   start .int_set:N = \l_@@_start_int ,
5962   end .code:n =
5963     \bool_lazy_or:nnTF
5964     { \tl_if_empty_p:n { #1 } }
5965     { \str_if_eq_p:nn { #1 } { last } }
5966     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5967     { \int_set:Nn \l_@@_end_int { #1 } }
5968 }
```

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.

```

5969 \keys_define:nn { NiceMatrix / RulesBis }
5970 {
5971   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5972   multiplicity .initial:n = 1 ,
5973   dotted .bool_set:N = \l_@@_dotted_bool ,
5974   dotted .initial:n = false ,
5975   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]`.

```

5976   color .code:n =
5977     \@@_set_CT@arc@:n { #1 }
5978     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5979   color .value_required:n = true ,
5980   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5981   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.

```

5982   tikz .code:n =
5983     \IfPackageLoadedTF { tikz }
5984       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5985       { \@@_error:n { tikz~without~tikz } } ,
5986   tikz .value_required:n = true ,
5987   total-width .dim_set:N = \l_@@_rule_width_dim ,
5988   total-width .value_required:n = true ,
5989   width .meta:n = { total-width = #1 } ,
5990   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5991 }

```

The vertical rules

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

```

5992 \cs_new_protected:Npn \@@_vline:n #1
5993 {

```

The group is for the options.

```

5994   \group_begin:
5995   \int_set_eq:NN \l_@@_end_int \c@iRow
5996   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

5997   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5998   \@@_vline_i:
5999   \group_end:
6000 }
6001 \cs_new_protected:Npn \@@_vline_i:
6002 {

```

`\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`.

```

6003   \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6004   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6005     \l_tmpa_tl
6006   {

```

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.

```

6007     \bool_gset_true:N \g_tmpa_bool
6008     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6009       { \@@_test_vline_in_block:nnnnn ##1 }
6010     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6011       { \@@_test_vline_in_block:nnnnn ##1 }
6012     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6013       { \@@_test_vline_in_stroken_block:nnnn ##1 }
6014     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6015     \bool_if:NTF \g_tmpa_bool
6016       {
6017         \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.

```

6018         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6019       }
6020     {
6021       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6022       {
6023         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6024         \@@_vline_ii:
6025         \int_zero:N \l_@@_local_start_int
6026       }
6027     }
6028   }
6029   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6030   {
6031     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6032     \@@_vline_ii:
6033   }
6034 }

```

```

6035 \cs_new_protected:Npn \@@_test_in_corner_v:
6036 {
6037   \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6038   {
6039     \seq_if_in:NxT
6040       \l_@@_corners_cells_seq
6041       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6042     { \bool_set_false:N \g_tmpa_bool }
6043   }
6044   {
6045     \seq_if_in:NxT
6046       \l_@@_corners_cells_seq
6047       { \l_tmpa_tl - \l_tmpb_tl }
6048     {
6049       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6050       { \bool_set_false:N \g_tmpa_bool }
6051       {
6052         \seq_if_in:NxT
6053           \l_@@_corners_cells_seq
6054           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6055         { \bool_set_false:N \g_tmpa_bool }
6056       }
6057     }
6058   }
6059 }

```

```

6060 \cs_new_protected:Npn \@@_vline_ii:
6061 {
6062   \tl_clear:N \l_@@_tikz_rule_tl
6063   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6064   \bool_if:NTF \l_@@_dotted_bool
6065     \@@_vline_iv:
6066     {
6067       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6068         \@@_vline_iii:
6069         \@@_vline_v:
6070     }
6071 }

```

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

```

6072 \cs_new_protected:Npn \@@_vline_iii:
6073 {
6074   \pgfpicture
6075   \pgfrememberpicturepositiononpagetrue
6076   \pgf@relevantforpicturesizefalse
6077   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6078   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6079   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6080   \dim_set:Nn \l_tmpb_dim
6081   {
6082     \pgf@x
6083     - 0.5 \l_@@_rule_width_dim
6084     +
6085     ( \arrayrulewidth * \l_@@_multiplicity_int
6086       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6087   }
6088   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6089   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6090   \bool_lazy_all:nT
6091   {
6092     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6093     { \cs_if_exist_p:N \CT@drsc@ }
6094     { ! \tl_if_blank_p:o \CT@drsc@ }
6095   }
6096   {
6097     \group_begin:
6098     \CT@drsc@
6099     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6100     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6101     \dim_set:Nn \l_@@_tmpd_dim
6102     {
6103       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6104       * ( \l_@@_multiplicity_int - 1 )
6105     }
6106     \pgfpathrectanglecorners
6107     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6108     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6109     \pgfusepath { fill }
6110     \group_end:
6111   }
6112   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6113   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6114   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6115   {
6116     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6117     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6118     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6119     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6120   }

```



```

6121 \CT@arc@
6122 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6123 \pgfsetrectcap
6124 \pgfusepathqstroke
6125 \endpgfpicture
6126 }

```

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

```

6127 \cs_new_protected:Npn \@@_vline_iv:
6128 {
6129 \pgfpicture
6130 \pgfrememberpicturepositiononpagetrue
6131 \pgf@relevantforpicturesizefalse
6132 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6133 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6134 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6135 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6136 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6137 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6138 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6139 \CT@arc@
6140 \@@_draw_line:
6141 \endpgfpicture
6142 }

```

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

```

6143 \cs_new_protected:Npn \@@_vline_v:
6144 {
6145 \begin {tikzpicture }

```

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).

```

6146 \CT@arc@
6147 \tl_if_empty:NF \l_@@_rule_color_tl
6148 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6149 \pgfrememberpicturepositiononpagetrue
6150 \pgf@relevantforpicturesizefalse
6151 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6152 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6153 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6154 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6155 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6156 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6157 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6158 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6159 ( \l_tmpb_dim , \l_tmpa_dim ) --
6160 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6161 \end { tikzpicture }
6162 }

```

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).

```

6163 \cs_new_protected:Npn \@@_draw_vlines:
6164 {
6165 \int_step_inline:nnn
6166 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6167 {
6168 \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6169 \c@jCol
6170 { \int_eval:n { \c@jCol + 1 } }
6171 }

```

```

6172 {
6173   \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6174   { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6175   { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6176 }
6177 }

```

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}`.

```

6178 \cs_new_protected:Npn \@@_hline:n #1
6179 {

```

The group is for the options.

```

6180   \group_begin:
6181   \int_zero_new:N \l_@@_end_int
6182   \int_set_eq:NN \l_@@_end_int \c@jCol
6183   \keys_set_known:nN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6184   \@@_hline_i:
6185   \group_end:
6186 }

6187 \cs_new_protected:Npn \@@_hline_i:
6188 {
6189   \int_zero_new:N \l_@@_local_start_int
6190   \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`.

```

6191   \tl_set:Nn \l_tmpa_tl { \int_use:N \l_@@_position_int }
6192   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6193   \l_tmpb_tl
6194   {

```

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.

```

6195     \bool_gset_true:N \g_tmpa_bool
6196     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6197     { \@@_test_hline_in_block:nnnnn ##1 }
6198     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6199     { \@@_test_hline_in_block:nnnnn ##1 }
6200     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6201     { \@@_test_hline_in_stroken_block:nnnn ##1 }
6202     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6203     \bool_if:NTF \g_tmpa_bool
6204     {
6205       \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.

```

6206       { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6207     }
6208     {
6209       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6210       {
6211         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6212         \@@_hline_ii:
6213         \int_zero:N \l_@@_local_start_int
6214       }
6215     }
6216   }

```

```

6217 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6218 {
6219     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6220     \@@_hline_ii:
6221 }
6222 }

6223 \cs_new_protected:Npn \@@_test_in_corner_h:
6224 {
6225     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6226     {
6227         \seq_if_in:NxT
6228         \l_@@_corners_cells_seq
6229         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6230         { \bool_set_false:N \g_tmpa_bool }
6231     }
6232     {
6233         \seq_if_in:NxT
6234         \l_@@_corners_cells_seq
6235         { \l_tmpa_tl - \l_tmpb_tl }
6236         {
6237             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6238             { \bool_set_false:N \g_tmpa_bool }
6239             {
6240                 \seq_if_in:NxT
6241                 \l_@@_corners_cells_seq
6242                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6243                 { \bool_set_false:N \g_tmpa_bool }
6244             }
6245         }
6246     }
6247 }

6248 \cs_new_protected:Npn \@@_hline_ii:
6249 {
6250     \tl_clear:N \l_@@_tikz_rule_tl
6251     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6252     \bool_if:NTF \l_@@_dotted_bool
6253     \@@_hline_iv:
6254     {
6255         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6256         \@@_hline_iii:
6257         \@@_hline_v:
6258     }
6259 }

```

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

```

6260 \cs_new_protected:Npn \@@_hline_iii:
6261 {
6262     \pgfpicture
6263     \pgfrememberpicturepositiononpagetrue
6264     \pgf@relevantforpicturesizefalse
6265     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6266     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6267     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6268     \dim_set:Nn \l_tmpb_dim
6269     {
6270         \pgf@y
6271         - 0.5 \l_@@_rule_width_dim
6272         +
6273         ( \arrayrulewidth * \l_@@_multiplicity_int

```

```

6274         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6275     }
6276     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6277     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6278     \bool_lazy_all:nT
6279     {
6280         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6281         { \cs_if_exist_p:N \CT@drsc@ }
6282         { ! \tl_if_blank_p:o \CT@drsc@ }
6283     }
6284     {
6285         \group_begin:
6286         \CT@drsc@
6287         \dim_set:Nn \l_@@_tmpd_dim
6288         {
6289             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6290             * ( \l_@@_multiplicity_int - 1 )
6291         }
6292         \pgfpathrectanglecorners
6293         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6294         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6295         \pgfusepathqfill
6296         \group_end:
6297     }
6298     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6299     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6300     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6301     {
6302         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6303         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6304         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6305         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6306     }
6307     \CT@arc@
6308     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6309     \pgfsetrectcap
6310     \pgfusepathqstroke
6311     \endpgfpicture
6312 }

```

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}

```

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

```

6313 \cs_new_protected:Npn \@@_hline_iv:
6314 {
6315     \pgfpicture
6316     \pgfrememberpicturepositiononpagetrue

```

```

6317 \pgf@relevantforpicturesizefalse
6318 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6319 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6320 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6321 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6322 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6323 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6324 {
6325   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6326   \bool_if:NF \g_@@_delims_bool
6327   { \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.

```

6328   \tl_if_eq:NnF \g_@@_left_delim_tl (
6329     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6330   )
6331 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6332 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6333 \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6334 {
6335   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6336   \bool_if:NF \g_@@_delims_bool
6337   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6338   \tl_if_eq:NnF \g_@@_right_delim_tl )
6339   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6340 }
6341 \CT@arc@
6342 \@@_draw_line:
6343 \endpgfpicture
6344 }

```

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`).

```

6345 \cs_new_protected:Npn \@@_hline_v:
6346 {
6347   \begin { tikzpicture }

```

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).

```

6348 \CT@arc@
6349 \tl_if_empty:NF \l_@@_rule_color_tl
6350 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6351 \pgfrememberpicturepositiononpagetrue
6352 \pgf@relevantforpicturesizefalse
6353 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6354 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6355 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6356 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6357 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6358 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6359 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6360 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6361 ( \l_tmpa_dim , \l_tmpb_dim ) --
6362 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6363 \end { tikzpicture }
6364 }

```

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).

```

6365 \cs_new_protected:Npn \@@_draw_hlines:
6366 {
6367   \int_step_inline:nnn
6368     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6369     {
6370       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6371       \c@iRow
6372       { \int_eval:n { \c@iRow + 1 } }
6373     }
6374     {
6375       \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6376       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6377       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6378     }
6379   }

```

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

```

6380 \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.

```

6381 \cs_set:Npn \@@_Hline_i:n #1
6382 {
6383   \peek_remove_spaces:n
6384   {
6385     \peek_meaning:NTF \Hline
6386     { \@@_Hline_ii:nn { #1 + 1 } }
6387     { \@@_Hline_iii:n { #1 } }
6388   }
6389 }
6390 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6391 \cs_set:Npn \@@_Hline_iii:n #1
6392 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6393 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6394 {
6395   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6396   \skip_vertical:N \l_@@_rule_width_dim
6397   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6398   {
6399     \@@_hline:n
6400     {
6401       multiplicity = #1 ,
6402       position = \int_eval:n { \c@iRow + 1 } ,
6403       total-width = \dim_use:N \l_@@_rule_width_dim ,
6404       #2
6405     }
6406   }
6407   \egroup
6408 }

```

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`).

```

6409 \cs_new_protected:Npn \@@_custom_line:n #1
6410 {
6411   \str_clear_new:N \l_@@_command_str
6412   \str_clear_new:N \l_@@_ccommand_str
6413   \str_clear_new:N \l_@@_letter_str
6414   \tl_clear_new:N \l_@@_other_keys_tl
6415   \keys_set_known:nnN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6416 \bool_lazy_all:nTF
6417 {
6418   { \str_if_empty_p:N \l_@@_letter_str }
6419   { \str_if_empty_p:N \l_@@_command_str }
6420   { \str_if_empty_p:N \l_@@_ccommand_str }
6421 }
6422 { \@@_error:n { No~letter~and~no~command } }
6423 { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6424 }
6425 \keys_define:nn { NiceMatrix / custom-line }
6426 {
6427   letter .str_set:N = \l_@@_letter_str ,
6428   letter .value_required:n = true ,
6429   command .str_set:N = \l_@@_command_str ,
6430   command .value_required:n = true ,
6431   ccommand .str_set:N = \l_@@_ccommand_str ,
6432   ccommand .value_required:n = true ,
6433 }
6434 \cs_new_protected:Npn \@@_custom_line_i:n #1
6435 {

```

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

```

6436 \bool_set_false:N \l_@@_tikz_rule_bool
6437 \bool_set_false:N \l_@@_dotted_rule_bool
6438 \bool_set_false:N \l_@@_color_bool
6439 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6440 \bool_if:NT \l_@@_tikz_rule_bool
6441 {
6442   \IfPackageLoadedTF { tikz }
6443   { }
6444   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6445   \bool_if:NT \l_@@_color_bool
6446   { \@@_error:n { color~in~custom~line~with~tikz } }
6447 }
6448 \bool_if:NT \l_@@_dotted_rule_bool
6449 {
6450   \int_compare:nNt \l_@@_multiplicity_int > \c_one_int
6451   { \@@_error:n { key~multiplicity~with~dotted } }
6452 }
6453 \str_if_empty:NF \l_@@_letter_str
6454 {
6455   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6456   { \@@_error:n { Several~letters } }
6457   {
6458     \exp_args:NnV \tl_if_in:NnTF
6459     \c_@@_forbidden_letters_str \l_@@_letter_str
6460     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6461     {

```

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.

```

6462 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6463 { \@@_v_custom_line:n { #1 } }
6464 }

```

```

6465     }
6466   }
6467   \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6468   \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6469 }
6470 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6471 \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.

```

6472 \keys_define:nn { NiceMatrix / custom-line-bis }
6473 {
6474   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6475   multiplicity .initial:n = 1 ,
6476   multiplicity .value_required:n = true ,
6477   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6478   color .value_required:n = true ,
6479   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6480   tikz .value_required:n = true ,
6481   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6482   dotted .value_forbidden:n = true ,
6483   total-width .code:n = { } ,
6484   total-width .value_required:n = true ,
6485   width .code:n = { } ,
6486   width .value_required:n = true ,
6487   sep-color .code:n = { } ,
6488   sep-color .value_required:n = true ,
6489   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6490 }

```

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

```

6491 \bool_new:N \l_@@_dotted_rule_bool
6492 \bool_new:N \l_@@_tikz_rule_bool
6493 \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`.

```

6494 \keys_define:nn { NiceMatrix / custom-line-width }
6495 {
6496   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6497   multiplicity .initial:n = 1 ,
6498   multiplicity .value_required:n = true ,
6499   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6500   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6501                       \bool_set_true:N \l_@@_total_width_bool ,
6502   total-width .value_required:n = true ,
6503   width .meta:n = { total-width = #1 } ,
6504   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6505 }

```

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`).

```

6506 \cs_new_protected:Npn \@@_h_custom_line:n #1
6507 {

```


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`).

```

6508     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6509     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6510 }

```

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`).

```

6511 \cs_new_protected:Npn \@@_c_custom_line:n #1
6512 {

```

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

```

6513     \exp_args:Nc \NewExpandableDocumentCommand
6514     { nicematrix - \l_@@_ccommand_str }
6515     { 0 { } m }
6516     {
6517         \noalign
6518         {
6519             \@@_compute_rule_width:n { #1 , ##1 }
6520             \skip_vertical:n { \l_@@_rule_width_dim }
6521             \clist_map_inline:nn
6522             { ##2 }
6523             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6524         }
6525     }
6526     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6527 }

```

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*.

```

6528 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6529 {
6530     \str_if_in:nnTF { #2 } { - }
6531     { \@@_cut_on_hyphen:w #2 \q_stop }
6532     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6533     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6534     {
6535         \@@_hline:n
6536         {
6537             #1 ,
6538             start = \l_tmpa_tl ,
6539             end = \l_tmpb_tl ,
6540             position = \int_eval:n { \c@iRow + 1 } ,
6541             total-width = \dim_use:N \l_@@_rule_width_dim
6542         }
6543     }
6544 }

6545 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6546 {
6547     \bool_set_false:N \l_@@_tikz_rule_bool
6548     \bool_set_false:N \l_@@_total_width_bool
6549     \bool_set_false:N \l_@@_dotted_rule_bool
6550     \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6551     \bool_if:NF \l_@@_total_width_bool
6552     {
6553         \bool_if:NTF \l_@@_dotted_rule_bool
6554         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6555         {
6556             \bool_if:NF \l_@@_tikz_rule_bool
6557             {

```

```

6558         \dim_set:Nn \l_@@_rule_width_dim
6559         {
6560             \arrayrulewidth * \l_@@_multiplicity_int
6561             + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6562         }
6563     }
6564 }
6565 }
6566 }
6567 \cs_new_protected:Npn \@@_v_custom_line:n #1
6568 {
6569     \@@_compute_rule_width:n { #1 }

```

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

```

6570     \tl_gput_right:Nx \g_@@_array_preamble_tl
6571     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6572     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6573     {
6574         \@@_vline:n
6575         {
6576             #1 ,
6577             position = \int_eval:n { \c@jCol + 1 } ,
6578             total-width = \dim_use:N \l_@@_rule_width_dim
6579         }
6580     }
6581     \@@_rec_preamble:n
6582 }
6583 \@@_custom_line:n
6584 { 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.

```

6585 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6586 {
6587     \int_compare:nNnT \l_tmpa_tl > { #1 }
6588     {
6589         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6590         {
6591             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6592             {
6593                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6594                 { \bool_gset_false:N \g_tmpa_bool }
6595             }
6596         }
6597     }
6598 }

```

The same for vertical rules.

```

6599 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6600 {
6601     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6602     {
6603         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6604         {
6605             \int_compare:nNnT \l_tmpb_tl > { #2 }
6606             {
6607                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6608                 { \bool_gset_false:N \g_tmpa_bool }
6609             }
6610         }
6611     }
6612 }

```

```

6610     }
6611   }
6612 }
6613 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6614 {
6615   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6616   {
6617     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6618     {
6619       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6620       { \bool_gset_false:N \g_tmpa_bool }
6621       {
6622         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6623         { \bool_gset_false:N \g_tmpa_bool }
6624       }
6625     }
6626   }
6627 }
6628 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6629 {
6630   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6631   {
6632     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6633     {
6634       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6635       { \bool_gset_false:N \g_tmpa_bool }
6636       {
6637         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6638         { \bool_gset_false:N \g_tmpa_bool }
6639       }
6640     }
6641   }
6642 }

```

24 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```

6643 \cs_new_protected:Npn \@@_compute_corners:
6644 {

```

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.

```

6645   \seq_clear_new:N \l_@@_corners_cells_seq
6646   \clist_map_inline:Nn \l_@@_corners_clist
6647   {
6648     \str_case:nnF { ##1 }
6649     {
6650       { NW }
6651       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6652       { NE }
6653       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6654       { SW }
6655       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6656       { SE }
6657       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6658     }

```

```

6659         { \@@_error:nn { bad-corner } { ##1 } }
6660     }

```

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

```

6661     \seq_if_empty:NF \l_@@_corners_cells_seq
6662     {

```

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.

```

6663         \tl_gput_right:Nx \g_@@_aux_tl
6664         {
6665             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6666             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6667         }
6668     }
6669 }

```

“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.

```

6670 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6671 {

```

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.

```

6672     \bool_set_false:N \l_tmpa_bool
6673     \int_zero_new:N \l_@@_last_empty_row_int
6674     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6675     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6676     {
6677         \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6678         \bool_lazy_or:nnTF
6679         {
6680             \cs_if_exist_p:c
6681             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6682         }
6683         \l_tmpb_bool
6684         { \bool_set_true:N \l_tmpa_bool }
6685         {
6686             \bool_if:NF \l_tmpa_bool
6687             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6688         }
6689     }

```

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

```

6690     \bool_set_false:N \l_tmpa_bool
6691     \int_zero_new:N \l_@@_last_empty_column_int
6692     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6693     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6694     {
6695         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }

```

```

6696 \bool_lazy_or:nnTF
6697 \l_tmpb_bool
6698 {
6699   \cs_if_exist_p:c
6700   { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6701 }
6702 { \bool_set_true:N \l_tmpa_bool }
6703 {
6704   \bool_if:NF \l_tmpa_bool
6705   { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6706 }
6707 }

```

Now, we loop over the rows.

```

6708 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6709 {

```

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

```

6710 \bool_set_false:N \l_tmpa_bool
6711 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6712 {
6713   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6714   \bool_lazy_or:nnTF
6715   \l_tmpb_bool
6716   {
6717     \cs_if_exist_p:c
6718     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6719   }
6720   { \bool_set_true:N \l_tmpa_bool }
6721   {
6722     \bool_if:NF \l_tmpa_bool
6723     {
6724       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6725       \seq_put_right:Nn
6726       \l_@@_corners_cells_seq
6727       { ##1 - #####1 }
6728     }
6729   }
6730 }
6731 }
6732 }

```

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`).

```

6733 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6734 {
6735   \int_set:Nn \l_tmpa_int { #1 }
6736   \int_set:Nn \l_tmpb_int { #2 }
6737   \bool_set_false:N \l_tmpb_bool
6738   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6739   { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6740 }
6741 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6 #7
6742 {
6743   \int_compare:nNnF { #3 } > { #1 }
6744   {
6745     \int_compare:nNnF { #1 } > { #5 }
6746     {
6747       \int_compare:nNnF { #4 } > { #2 }
6748       {
6749         \int_compare:nNnF { #2 } > { #6 }
6750         { \bool_set_true:N \l_tmpb_bool }

```

```

6751     }
6752   }
6753 }
6754 }

```

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.

```

6755 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6756 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6757 {
6758   auto-columns-width .code:n =
6759   {
6760     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6761     \dim_gzero_new:N \g_@@_max_cell_width_dim
6762     \bool_set_true:N \l_@@_auto_columns_width_bool
6763   }
6764 }

6765 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6766 {
6767   \int_gincr:N \g_@@_NiceMatrixBlock_int
6768   \dim_zero:N \l_@@_columns_width_dim
6769   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6770   \bool_if:NT \l_@@_block_auto_columns_width_bool
6771   {
6772     \cs_if_exist:cT
6773     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6774     {
6775       % is \exp_args:NNe mandatory?
6776       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6777       {
6778         \use:c
6779         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6780       }
6781     }
6782   }
6783 }

```

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).

```

6784 {
6785   \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.

```

6786   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6787   {
6788     \bool_if:NT \l_@@_block_auto_columns_width_bool
6789     {
6790       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6791       \iow_shipout:Nx \@mainaux
6792       {
6793         \cs_gset:cpn
6794         { @@ _ 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.

```

6795         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6796     }
6797     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6798 }
6799 }
6800 \ignorespacesafterend
6801 }

```

26 The extra nodes

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

```

6802 \cs_generate_variant:Nn \dim_min:nn { v n }
6803 \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).

```

6804 \cs_new_protected:Npn \@@_create_extra_nodes:
6805 {
6806     \bool_if:nTF \l_@@_medium_nodes_bool
6807     {
6808         \bool_if:NTF \l_@@_large_nodes_bool
6809         \@@_create_medium_and_large_nodes:
6810         \@@_create_medium_nodes:
6811     }
6812     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6813 }

```

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`.

```

6814 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6815 {
6816     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6817     {
6818         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6819         \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6820         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6821         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6822     }
6823     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6824     {

```

```

6825     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6826     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6827     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6828     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6829 }

```

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

```

6830     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6831     {
6832         \int_step_variable:nnNn
6833         \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.

```

6834         {
6835             \cs_if_exist:cT
6836             { 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`.

```

6837         {
6838             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6839             \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6840             { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6841             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6842             {
6843                 \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6844                 { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6845             }

```

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`.

```

6846             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6847             \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6848             { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6849             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6850             {
6851                 \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6852                 { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6853             }
6854         }
6855     }
6856 }

```

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

```

6857     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6858     {
6859         \dim_compare:nNnT
6860         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6861         {
6862             \@@_qpoint:n { row - \@@_i: - base }
6863             \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6864             \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6865         }
6866     }
6867     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6868     {
6869         \dim_compare:nNnT
6870         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6871         {
6872             \@@_qpoint:n { col - \@@_j: }
6873             \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6874             \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6875         }

```



```

6876     }
6877 }

```

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

```

6878 \cs_new_protected:Npn \@@_create_medium_nodes:
6879 {
6880   \pgfpicture
6881     \pgfrememberpicturepositiononpagetrue
6882     \pgf@relevantforpicturesizefalse
6883     \@@_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”.

```

6884     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6885     \@@_create_nodes:
6886   \endpgfpicture
6887 }

```

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:`.

```

6888 \cs_new_protected:Npn \@@_create_large_nodes:
6889 {
6890   \pgfpicture
6891     \pgfrememberpicturepositiononpagetrue
6892     \pgf@relevantforpicturesizefalse
6893     \@@_computations_for_medium_nodes:
6894     \@@_computations_for_large_nodes:
6895     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6896     \@@_create_nodes:
6897   \endpgfpicture
6898 }
6899 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6900 {
6901   \pgfpicture
6902     \pgfrememberpicturepositiononpagetrue
6903     \pgf@relevantforpicturesizefalse
6904     \@@_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”.

```

6905     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6906     \@@_create_nodes:
6907     \@@_computations_for_large_nodes:
6908     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6909     \@@_create_nodes:
6910   \endpgfpicture
6911 }

```

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.

```

6912 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6913 {
6914   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6915   \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

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

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`.

```

6916 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6917 {
6918   \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6919   {
6920     (
6921       \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6922       \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6923     )
6924     / 2
6925   }
6926   \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6927   { l_@@_row _ \@@_i: _ min _ dim }
6928 }
6929 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6930 {
6931   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6932   {
6933     (
6934       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6935       \dim_use:c
6936       { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6937     )
6938     / 2
6939   }
6940   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6941   { l_@@_column _ \@@_j: _ max _ dim }
6942 }

```

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

```

6943 \dim_sub:cn
6944 { l_@@_column _ 1 _ min _ dim }
6945 \l_@@_left_margin_dim
6946 \dim_add:cn
6947 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6948 \l_@@_right_margin_dim
6949 }

```

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).

```

6950 \cs_new_protected:Npn \@@_create_nodes:
6951 {
6952   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6953   {
6954     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6955     {

```

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

```

6956 \@@_pgf_rect_node:nnnnn
6957 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6958 { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6959 { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6960 { \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } }
6961 { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6962 \str_if_empty:NF \l_@@_name_str
6963 {
6964   \pgfnodealias
6965   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }

```

```

6966         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6967     }
6968 }
6969 }

```

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 .

```

6970     \seq_map_pairwise_function:NNN
6971     \g_@@_multicolumn_cells_seq
6972     \g_@@_multicolumn_sizes_seq
6973     \@@_node_for_multicolumn:nn
6974 }

```

```

6975 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6976 {
6977     \cs_set_nopar:Npn \@@_i: { #1 }
6978     \cs_set_nopar:Npn \@@_j: { #2 }
6979 }

```

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”).

```

6980 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6981 {
6982     \@@_extract_coords_values: #1 \q_stop
6983     \@@_pgf_rect_node:nnnnn
6984     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6985     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6986     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6987     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
6988     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6989     \str_if_empty:NF \l_@@_name_str
6990     {
6991         \pgfnodealias
6992         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6993         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6994     }
6995 }

```

27 The blocks

The following 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 (in the cell of the array).

```

6996 \keys_define:nn { NiceMatrix / Block / FirstPass }
6997 {
6998     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6999     l .value_forbidden:n = true ,
7000     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7001     r .value_forbidden:n = true ,
7002     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7003     c .value_forbidden:n = true ,
7004     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7005     L .value_forbidden:n = true ,

```

```

7006 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7007 R .value_forbidden:n = true ,
7008 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7009 C .value_forbidden:n = true ,
7010 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7011 t .value_forbidden:n = true ,
7012 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7013 T .value_forbidden:n = true ,
7014 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7015 b .value_forbidden:n = true ,
7016 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7017 B .value_forbidden:n = true ,
7018 color .code:n =
7019   \@@_color:n { #1 }
7020   \tl_set_rescan:Nnn
7021     \l_@@_draw_tl
7022     { \char_set_catcode_other:N ! }
7023     { #1 } ,
7024 color .value_required:n = true ,
7025 respect-arraystretch .code:n =
7026   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7027 respect-arraystretch .value_forbidden:n = true ,
7028 }

```

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.

```

7029 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7030 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7031 {

```

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).

```

7032   \peek_remove_spaces:n
7033   {
7034     \tl_if_blank:nTF { #2 }
7035     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7036     {
7037       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7038       \@@_Block_i_czech \@@_Block_i
7039       #2 \q_stop
7040     }
7041     { #1 } { #3 } { #4 }
7042   }
7043 }

```

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

```

7044 \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`.

```

7045 {
7046   \char_set_catcode_active:N -
7047   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7048 }

```

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.

```

7049 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7050 {

```

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).

```

7051 \bool_lazy_or:nnTF
7052 { \tl_if_blank_p:n { #1 } }
7053 { \str_if_eq_p:nn { #1 } { * } }
7054 { \int_set:Nn \l_tmpa_int { 100 } }
7055 { \int_set:Nn \l_tmpa_int { #1 } }
7056 \bool_lazy_or:nnTF
7057 { \tl_if_blank_p:n { #2 } }
7058 { \str_if_eq_p:nn { #2 } { * } }
7059 { \int_set:Nn \l_tmpb_int { 100 } }
7060 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7061 \int_compare:nNnTF \l_tmpb_int = \c_one_int
7062 {
7063   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7064     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7065     { \str_set:NW \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7066 }
7067 { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }

```

The value of \l_@@_hpos_block_str may be modified by the keys of the command \Block that we will analyze now.

```

7068 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
7069 \tl_set:Nx \l_tmpa_tl
7070 {
7071   { \int_use:N \c@iRow }
7072   { \int_use:N \c@jCol }
7073   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7074   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7075 }

```

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).

```

7076 \bool_if:nTF
7077 {
7078   (
7079     \int_compare_p:nNn \l_tmpa_int = \c_one_int
7080     ||
7081     \int_compare_p:nNn \l_tmpb_int = \c_one_int
7082   )
7083   && ! \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.

```

7084 && ! \l_@@_X_bool
7085 }
7086 { \exp_args:Nee \@@_Block_iv:nnnnn }
7087 { \exp_args:Nee \@@_Block_v:nnnnn }
7088 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7089 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both). In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

`#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of `key=values` pairs, `#4` are the tokens to put before the potential math mode and before the composition of the block and `#5` is the label (=content) of the block.

```

7090 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7091 {
7092   \int_gincr:N \g_@@_block_box_int
7093   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7094   {
7095     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7096     {
7097       \@@_actually_diagbox:nnnnnn
7098       { \int_use:N \c@iRow }
7099       { \int_use:N \c@jCol }
7100       { \int_eval:n { \c@iRow + #1 - 1 } }
7101       { \int_eval:n { \c@jCol + #2 - 1 } }
7102       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7103       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7104     }
7105   }
7106   \box_gclear_new:c
7107   { \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!

```

7108   \hbox_gset:cn
7109   { \g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7110   {

```

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}`).

```

7111   \tl_if_empty:NTF \l_@@_color_tl
7112   { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7113   { \@@_color:o \l_@@_color_tl }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

7114   \int_compare:nNnT { #1 } = \c_one_int
7115   {
7116     \int_if_zero:nTF \c@iRow
7117     \l_@@_code_for_first_row_tl
7118     {
7119       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7120       \l_@@_code_for_last_row_tl
7121     }
7122     \g_@@_row_style_tl
7123   }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7124   \@@_reset_arraystretch:
7125   \dim_zero:N \extrarowheight

```

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

```

7126   #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.).

```
7127 \@@_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}`.

```
7128 \bool_if:NTF \l_@@_tabular_bool
7129 {
7130   \bool_lazy_all:nTF
7131   {
7132     { \int_compare_p:nNn { #2 } = \c_one_int }
```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```
7133     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7134     { ! \g_@@_rotate_bool }
7135   }
```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```
7136   {
7137     \use:e
7138     {
7139       \exp_not:N \begin { minipage }%
7140       [ \str_lowercase:V \l_@@_vpos_block_str ]
7141       { \l_@@_col_width_dim }
7142       \str_case:on \l_@@_hpos_block_str
7143       { c \centering r \raggedleft l \raggedright }
7144     }
7145     #5
7146     \end { minipage }
7147   }
```

In the other cases, we use a `{tabular}`.

```
7148   {
7149     \use:e
7150     {
7151       \exp_not:N \begin { tabular }%
7152       [ \str_lowercase:V \l_@@_vpos_block_str ]
7153       { @ { } \l_@@_hpos_block_str @ { } }
7154     }
7155     #5
7156     \end { tabular }
7157   }
7158 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7159   {
7160     \c_math_toggle_token
7161     \use:e
7162     {
7163       \exp_not:N \begin { array }%
7164       [ \str_lowercase:V \l_@@_vpos_block_str ]
7165       { @ { } \l_@@_hpos_block_str @ { } }
7166     }
7167     #5
7168     \end { array }
7169     \c_math_toggle_token
7170   }
7171 }
```

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).

```
7172 \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.

```
7173 \int_compare:nNtT { #2 } = \c_one_int
7174 {
7175   \dim_gset:Nn \g_@@_blocks_wd_dim
7176   {
7177     \dim_max:nn
7178     \g_@@_blocks_wd_dim
7179     {
7180       \box_wd:c
7181       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7182     }
7183   }
7184 }
```

If we are in a mono-row block and if that block has no vertical option for the position¹⁵, we take into account the height and the depth of that block for the height and the depth of the row.

```
7185 \str_if_eq:VnT \l_@@_vpos_block_str { c }
7186 {
7187   \int_compare:nNtT { #1 } = \c_one_int
7188   {
7189     \dim_gset:Nn \g_@@_blocks_ht_dim
7190     {
7191       \dim_max:nn
7192       \g_@@_blocks_ht_dim
7193       {
7194         \box_ht:c
7195         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7196       }
7197     }
7198     \dim_gset:Nn \g_@@_blocks_dp_dim
7199     {
7200       \dim_max:nn
7201       \g_@@_blocks_dp_dim
7202       {
7203         \box_dp:c
7204         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7205       }
7206     }
7207   }
7208 }
7209 \seq_gput_right:Nx \g_@@_blocks_seq
7210 {
7211   \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.

```
7212 {
7213   \exp_not:n { #3 } ,
7214   \l_@@_hpos_block_str ,
```

¹⁵If the block has a key of a vertical position, that means that it has to be put in a vertical space determined by the *others* cells of the row. Therefore there is no point creating space here. Moreover, that would lead to problems when a multi-row block with a position key such as b or B.

Now, we put a key for the vertical alignment.

```

7215     \bool_if:NT \g_@@_rotate_bool
7216     {
7217         \bool_if:NTF \g_@@_rotate_c_bool
7218         { m }
7219         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7220     }
7221
7222 }
7223 {
7224     \box_use_drop:c
7225     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7226 }
7227 }
7228 \bool_set_false:N \g_@@_rotate_c_bool
7229 }

7230 \cs_new:Npn \@@_adjust_hpos_rotate:
7231 {
7232     \bool_if:NT \g_@@_rotate_bool
7233     {
7234         \str_set:Nx \l_@@_hpos_block_str
7235         {
7236             \bool_if:NTF \g_@@_rotate_c_bool
7237             { c }
7238             {
7239                 \str_case:onF \l_@@_vpos_block_str
7240                 { b l B l t r T r }
7241                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7242             }
7243         }
7244     }
7245 }
```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7246 \cs_new_protected:Npn \@@_rotate_box_of_block:
7247 {
7248     \box_grotate:cn
7249     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7250     { 90 }
7251     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7252     {
7253         \vbox_gset_top:cn
7254         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7255         {
7256             \skip_vertical:n { 0.8 ex }
7257             \box_use:c
7258             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7259         }
7260     }
7261     \bool_if:NT \g_@@_rotate_c_bool
7262     {
7263         \hbox_gset:cn
7264         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7265         {
7266             \c_math_toggle_token
7267             \vcenter
7268             {
7269                 \box_use:c
7270                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7271             }
7272         }
7273     }
7274 }
```

```

7272         \c_math_toggle_token
7273     }
7274 }
7275 }

```

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.

```

7276 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7277 {
7278     \seq_gput_right:Nx \g_@@_blocks_seq
7279     {
7280         \l_tmpa_tl
7281         { \exp_not:n { #3 } }
7282         {
7283             \bool_if:NTF \l_@@_tabular_bool
7284             {
7285                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7286         \@@_reset_arraystretch:
7287         \exp_not:n
7288         {
7289             \dim_zero:N \extrarowheight
7290             #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).

```

7291         \use:e
7292         {
7293             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7294             { @ { } \l_@@_hpos_block_str @ { } }
7295         }
7296         #5
7297         \end { tabular }
7298     }
7299     \group_end:
7300 }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7301     {
7302         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7303         \@@_reset_arraystretch:
7304         \exp_not:n
7305         {
7306             \dim_zero:N \extrarowheight
7307             #4
7308             \c_math_toggle_token
7309             \use:e
7310             {
7311                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7312                 { @ { } \l_@@_hpos_block_str @ { } }
7313             }
7314             #5
7315             \end { array }

```

```

7316         \c_math_toggle_token
7317     }
7318     \group_end:
7319 }
7320 }
7321 }
7322 }

```

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).

```

7323 \keys_define:nn { NiceMatrix / Block / SecondPass }
7324 {
7325     tikz .code:n =
7326         \IfPackageLoadedTF { tikz }
7327         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7328         { \@@_error:n { tikz~key~without~tikz } } ,
7329     tikz .value_required:n = true ,
7330     fill .code:n =
7331         \tl_set_rescan:Nnn
7332         \l_@@_fill_tl
7333         { \char_set_catcode_other:N ! }
7334         { #1 } ,
7335     fill .value_required:n = true ,
7336     opacity .tl_set:N = \l_@@_opacity_tl ,
7337     opacity .value_required:n = true ,
7338     draw .code:n =
7339         \tl_set_rescan:Nnn
7340         \l_@@_draw_tl
7341         { \char_set_catcode_other:N ! }
7342         { #1 } ,
7343     draw .default:n = default ,
7344     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7345     rounded-corners .default:n = 4 pt ,
7346     color .code:n =
7347         \@@_color:n { #1 }
7348         \tl_set_rescan:Nnn
7349         \l_@@_draw_tl
7350         { \char_set_catcode_other:N ! }
7351         { #1 } ,
7352     borders .clist_set:N = \l_@@_borders_clist ,
7353     borders .value_required:n = true ,
7354     hvlines .meta:n = { vlines , hlines } ,
7355     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7356     vlines .default:n = true ,
7357     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7358     hlines .default:n = true ,
7359     line-width .dim_set:N = \l_@@_line_width_dim ,
7360     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7361     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7362     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7363     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7364     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7365         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7366     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7367         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7368     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7369         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7370     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7371     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,

```

```

7372 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7373 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7374 m .code:n = \str_set:Nn \l_@@_vpos_block_str { c } ,
7375 m .value_forbidden:n = true ,
7376 v-center .meta:n = m ,
7377 name .tl_set:N = \l_@@_block_name_str ,
7378 name .value_required:n = true ,
7379 name .initial:n = ,
7380 respect-arraystretch .code:n =
7381   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7382 respect-arraystretch .value_forbidden:n = true ,
7383 transparent .bool_set:N = \l_@@_transparent_bool ,
7384 transparent .default:n = true ,
7385 transparent .initial:n = false ,
7386 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7387 }

```

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.

```

7388 \cs_new_protected:Npn \@@_draw_blocks:
7389 {
7390   \bool_if:NTF \c_@@_tagging_array_bool
7391     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7392     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7393   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7394 }
7395 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7396 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7397   \int_zero_new:N \l_@@_last_row_int
7398   \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.

```

7399   \int_compare:nNnTF { #3 } > { 99 }
7400     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7401     { \int_set:Nn \l_@@_last_row_int { #3 } }
7402   \int_compare:nNnTF { #4 } > { 99 }
7403     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7404     { \int_set:Nn \l_@@_last_col_int { #4 } }
7405   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7406     {
7407       \bool_lazy_and:nnTF
7408         \l_@@_preamble_bool
7409         {
7410           \int_compare_p:n
7411             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7412         }
7413         {
7414           \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7415           \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7416           \@@_msg_redirect_name:nn { columns-not-used } { none }
7417         }
7418       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7419     }

```

```

7420 {
7421   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7422     { \msg_error:nnnn { nicematrix } { Block-too~large~1 } { #1 } { #2 } }
7423     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7424 }
7425 }

```

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

```

7426 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7427 {

```

The group is for the keys.

```

7428   \group_begin:
7429   \int_compare:nNnT { #1 } = { #3 }
7430     { \str_set:Nn \l_@@_vpos_block_str { t } }
7431   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7432   \bool_if:NT \l_@@_vlines_block_bool
7433   {
7434     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7435     {
7436       \@@_vlines_block:nnn
7437       { \exp_not:n { #5 } }
7438       { #1 - #2 }
7439       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7440     }
7441   }
7442   \bool_if:NT \l_@@_hlines_block_bool
7443   {
7444     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7445     {
7446       \@@_hlines_block:nnn
7447       { \exp_not:n { #5 } }
7448       { #1 - #2 }
7449       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7450     }
7451   }
7452   \bool_if:NF \l_@@_transparent_bool
7453   {
7454     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7455     {

```

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).

```

7456       \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7457       { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7458     }
7459   }

```

```

7460   \tl_if_empty:NF \l_@@_draw_tl
7461   {
7462     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7463     { \@@_error:n { hlines-with~color } }
7464     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7465     {
7466       \@@_stroke_block:nnn

```

#5 are the options

```

7467       { \exp_not:n { #5 } }
7468       { #1 - #2 }
7469       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7470     }

```

```

7471     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7472     { { #1 } { #2 } { #3 } { #4 } }
7473 }
7474 \clist_if_empty:NF \l_@@_borders_clist
7475 {
7476     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7477     {
7478         \@@_stroke_borders_block:nnn
7479         { \exp_not:n { #5 } }
7480         { #1 - #2 }
7481         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7482     }
7483 }
7484 \tl_if_empty:NF \l_@@_fill_tl
7485 {
7486     \tl_if_empty:NF \l_@@_opacity_tl
7487     {
7488         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7489             {
7490                 \tl_set:Nx \l_@@_fill_tl
7491                 {
7492                     [ opacity = \l_@@_opacity_tl ,
7493                     \tl_tail:o \l_@@_fill_tl
7494                 }
7495             }
7496             {
7497                 \tl_set:Nx \l_@@_fill_tl
7498                 { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7499             }
7500         ]
7501         \tl_gput_right:Nx \g_@@_pre_code_before_tl
7502         {
7503             \exp_not:N \roundedrectanglecolor
7504             \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7505                 { \l_@@_fill_tl }
7506                 { { \l_@@_fill_tl } }
7507             { #1 - #2 }
7508             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7509             { \dim_use:N \l_@@_rounded_corners_dim }
7510         ]
7511     }
7512 \seq_if_empty:NF \l_@@_tikz_seq
7513 {
7514     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7515     {
7516         \@@_block_tikz:nnnnn
7517         { #1 }
7518         { #2 }
7519         { \int_use:N \l_@@_last_row_int }
7520         { \int_use:N \l_@@_last_col_int }
7521         { \seq_use:Nn \l_@@_tikz_seq { , } }
7522     }
7523 }
7524 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7525 {
7526     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7527     {
7528         \@@_actually_diagbox:nnnnnn
7529         { #1 }
7530         { #2 }
7531         { \int_use:N \l_@@_last_row_int }

```

```

7532         { \int_use:N \l_@@_last_col_int }
7533         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7534     }
7535 }

7536 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7537 \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.

```

7538 \pgfpicture
7539 \pgfrememberpicturepositiononpagetrue
7540 \pgf@relevantforpicturesizefalse
7541 \@@_qpoint:n { row - #1 }
7542 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7543 \@@_qpoint:n { col - #2 }
7544 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7545 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7546 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7547 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7548 \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.

```

7549 \@@_pgf_rect_node:nnnnn
7550 { \@@_env: - #1 - #2 - block }
7551 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7552 \str_if_empty:NF \l_@@_block_name_str
7553 {
7554     \pgfnodealias
7555     { \@@_env: - \l_@@_block_name_str }
7556     { \@@_env: - #1 - #2 - block }
7557     \str_if_empty:NF \l_@@_name_str
7558     {
7559         \pgfnodealias
7560         { \l_@@_name_str - \l_@@_block_name_str }
7561         { \@@_env: - #1 - #2 - block }
7562     }
7563 }

```

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.

```

7564     \bool_if:NF \l_@@_hpos_of_block_cap_bool
7565     {
7566         \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.

```

7567         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7568         {

```

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.

```

7569             \cs_if_exist:cT
7570             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7571             {
7572                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7573                 {
7574                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7575                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7576                 }
7577             }
7578         }

```

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.

```

7579         \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7580         {
7581             \@@_qpoint:n { col - #2 }
7582             \dim_set_eq:NN \l_tmpb_dim \pgf@x
7583         }
7584         \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7585         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7586         {
7587             \cs_if_exist:cT
7588             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7589             {
7590                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7591                 {
7592                     \pgfpointanchor
7593                     { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7594                     { east }
7595                     \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7596                 }
7597             }
7598         }
7599         \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7600         {
7601             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7602             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7603         }
7604         \@@_pgf_rect_node:nnnnn
7605         { \@@_env: - #1 - #2 - block - short }
7606         \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7607     }

```

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.

```

7608     \bool_if:NT \l_@@_medium_nodes_bool
7609     {
7610         \@@_pgf_rect_node:nnn
7611         { \@@_env: - #1 - #2 - block - medium }
7612         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7613         {

```



```

7614         \pgfpointanchor
7615         { \l_@@_env:
7616           - \int_use:N \l_@@_last_row_int
7617           - \int_use:N \l_@@_last_col_int - medium
7618         }
7619         { south-east }
7620     }
7621 }

```

Now, we will put the label of the block.

```

7622     \bool_lazy_any:nTF
7623     {
7624       { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7625       { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7626       { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7627     }
7628
7629     {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7629         \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the last column, we must put the block as if it was with the key l.

```

7630         \bool_if:nT \g_@@_last_col_found_bool
7631         {
7632           \int_compare:nNnT { #2 } = \g_@@_col_total_int
7633           { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7634         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7635     \tl_set:Nx \l_tmpa_tl
7636     {
7637       \str_case:on \l_@@_vpos_block_str
7638       {
7639         c {
7640           \str_case:on \l_@@_hpos_block_str
7641           {
7642             c { center }
7643             l { west }
7644             r { east }
7645           }
7646         }
7647       }
7648       T {
7649         \str_case:on \l_@@_hpos_block_str
7650         {
7651           c { north }
7652           l { north-west }
7653           r { north-east }
7654         }
7655       }
7656     }
7657     B {
7658       \str_case:on \l_@@_hpos_block_str
7659       {
7660         c { south }
7661         l { south-west }
7662         r { south-east }
7663       }
7664     }
7665   }
7666 }
7667

```

```

7668 \pgftransformshift
7669 {
7670   \pgfpointanchor
7671   {
7672     \@@_env: - #1 - #2 - block
7673     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7674   }
7675   { \l_tmpa_tl }
7676 }
7677 \pgfset
7678 {
7679   inner~xsep = \c_zero_dim ,
7680   inner~ysep = \c_zero_dim
7681 }
7682 \pgfnode
7683 { rectangle }
7684 { \l_tmpa_tl }
7685 { \box_use_drop:N \l_@@_cell_box } { } { }
7686 }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

7687 {
7688   \pgfextracty \l_tmpa_dim
7689   {
7690     \@@_qpoint:n
7691     {
7692       row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7693       - base
7694     }
7695   }
7696   \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

7697 \pgfpointanchor
7698 {
7699   \@@_env: - #1 - #2 - block
7700   \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7701 }
7702 {
7703   \str_case:on \l_@@_hpos_block_str
7704   {
7705     c { center }
7706     l { west }
7707     r { east }
7708   }
7709 }

```

We put the label of the block which has been composed in $\l_@@_cell_box$.

```

7710 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7711 \pgfset { inner~sep = \c_zero_dim }
7712 \pgfnode
7713 { rectangle }
7714 {
7715   \str_case:on \l_@@_hpos_block_str
7716   {
7717     c { base }
7718     l { base~west }
7719     r { base~east }
7720   }
7721 }
7722 { \box_use_drop:N \l_@@_cell_box } { } { }
7723 }

```

```

7724 \endpgfpicture
7725 \group_end:
7726 }

```

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).

```

7727 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7728 {
7729   \group_begin:
7730   \tl_clear:N \l_@@_draw_tl
7731   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7732   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7733   \pgfpicture
7734   \pgfrememberpicturepositiononpagetrue
7735   \pgf@relevantforpicturesizefalse
7736   \tl_if_empty:NF \l_@@_draw_tl
7737   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7738     \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7739     { \CT@arc@ }
7740     { \@@_color:o \l_@@_draw_tl }
7741   }
7742   \pgfsetcornersarced
7743   {
7744     \pgfpoint
7745     { \l_@@_rounded_corners_dim }
7746     { \l_@@_rounded_corners_dim }
7747   }
7748   \@@_cut_on_hyphen:w #2 \q_stop
7749   \int_compare:nNnF \l_tmpa_tl > \c@iRow
7750   {
7751     \int_compare:nNnF \l_tmpb_tl > \c@jCol
7752     {
7753       \@@_qpoint:n { row - \l_tmpa_tl }
7754       \dim_set_eq:NN \l_tmpb_dim \pgf@y
7755       \@@_qpoint:n { col - \l_tmpb_tl }
7756       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7757       \@@_cut_on_hyphen:w #3 \q_stop
7758       \int_compare:nNnT \l_tmpa_tl > \c@iRow
7759       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7760       \int_compare:nNnT \l_tmpb_tl > \c@jCol
7761       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7762       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7763       \dim_set_eq:NN \l_tmpa_dim \pgf@y
7764       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7765       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7766       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7767       \pgfpathrectanglecorners
7768       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7769       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7770       \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7771       { \pgfusepathqstroke }
7772       { \pgfusepath { stroke } }
7773     }
7774   }
7775   \endpgfpicture
7776   \group_end:
7777 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7778 \keys_define:nn { NiceMatrix / BlockStroke }
7779 {
7780   color .tl_set:N = \l_@@_draw_tl ,
7781   draw .code:n =
7782     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7783   draw .default:n = default ,
7784   line-width .dim_set:N = \l_@@_line_width_dim ,
7785   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7786   rounded-corners .default:n = 4 pt
7787 }

```

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).

```

7788 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7789 {
7790   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7791   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7792   \@@_cut_on_hyphen:w #2 \q_stop
7793   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7794   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7795   \@@_cut_on_hyphen:w #3 \q_stop
7796   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7797   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7798   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7799   {
7800     \use:e
7801     {
7802       \@@_vline:n
7803       {
7804         position = ##1 ,
7805         start = \l_@@_tmpc_tl ,
7806         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7807         total-width = \dim_use:N \l_@@_line_width_dim
7808       }
7809     }
7810   }
7811 }
7812 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7813 {
7814   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7815   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7816   \@@_cut_on_hyphen:w #2 \q_stop
7817   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7818   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7819   \@@_cut_on_hyphen:w #3 \q_stop
7820   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7821   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7822   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7823   {
7824     \use:e
7825     {
7826       \@@_hline:n
7827       {
7828         position = ##1 ,
7829         start = \l_@@_tmpd_tl ,
7830         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7831         total-width = \dim_use:N \l_@@_line_width_dim
7832       }
7833     }
7834   }
7835 }

```

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).

```

7836 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7837 {
7838   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7839   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7840   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7841     { \@@_error:n { borders~forbidden } }
7842     {
7843       \tl_clear_new:N \l_@@_borders_tikz_tl
7844       \keys_set:nV
7845         { NiceMatrix / OnlyForTikzInBorders }
7846         \l_@@_borders_clist
7847       \@@_cut_on_hyphen:w #2 \q_stop
7848       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7849       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7850       \@@_cut_on_hyphen:w #3 \q_stop
7851       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7852       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7853       \@@_stroke_borders_block_i:
7854     }
7855 }

7856 \hook_gput_code:nnn { begindocument } { . }
7857 {
7858   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7859   {
7860     \c_@@_pgfortikzpicture_tl
7861     \@@_stroke_borders_block_ii:
7862     \c_@@_endpgfortikzpicture_tl
7863   }
7864 }

7865 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7866 {
7867   \pgfrememberpicturerepositiononpagetrue
7868   \pgfrelevantforpicturesizefalse
7869   \CT@arc@
7870   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7871   \clist_if_in:NnT \l_@@_borders_clist { right }
7872     { \@@_stroke_vertical:n \l_tmpb_tl }
7873   \clist_if_in:NnT \l_@@_borders_clist { left }
7874     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7875   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7876     { \@@_stroke_horizontal:n \l_tmpa_tl }
7877   \clist_if_in:NnT \l_@@_borders_clist { top }
7878     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7879 }

7880 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7881 {
7882   tikz .code:n =
7883     \cs_if_exist:NTF \tikzpicture
7884       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7885       { \@@_error:n { tikz~in~borders~without~tikz } } ,
7886   tikz .value_required:n = true ,
7887   top .code:n = ,
7888   bottom .code:n = ,
7889   left .code:n = ,
7890   right .code:n = ,
7891   unknown .code:n = \@@_error:n { bad~border }
7892 }

```

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).

```

7893 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7894 {
7895   \@@_qpoint:n \l_@@_tmpc_tl
7896   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7897   \@@_qpoint:n \l_tmpa_tl
7898   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7899   \@@_qpoint:n { #1 }
7900   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7901   {
7902     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7903     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7904     \pgfusepath{stroke}
7905   }
7906   {
7907     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7908     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7909   }
7910 }

```

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).

```

7911 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7912 {
7913   \@@_qpoint:n \l_@@_tmpd_tl
7914   \clist_if_in:NnTF \l_@@_borders_clist { left }
7915   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7916   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7917   \@@_qpoint:n \l_tmpb_tl
7918   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7919   \@@_qpoint:n { #1 }
7920   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7921   {
7922     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7923     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7924     \pgfusepath{stroke}
7925   }
7926   {
7927     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7928     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7929   }
7930 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7931 \keys_define:nn { NiceMatrix / BlockBorders }
7932 {
7933   borders .clist_set:N = \l_@@_borders_clist ,
7934   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7935   rounded-corners .default:n = 4 pt ,
7936   line-width .dim_set:N = \l_@@_line_width_dim
7937 }

```

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.

```

7938 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7939 {
7940   \begin { tikzpicture }
7941   \@@_clip_with_rounded_corners:
7942   \clist_map_inline:nn { #5 }

```

```

7943 {
7944   \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7945   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7946   (
7947     [
7948       xshift = \dim_use:N \l_@@_offset_dim ,
7949       yshift = - \dim_use:N \l_@@_offset_dim
7950     ]
7951     #1 -| #2
7952   )
7953   rectangle
7954   (
7955     [
7956       xshift = - \dim_use:N \l_@@_offset_dim ,
7957       yshift = \dim_use:N \l_@@_offset_dim
7958     ]
7959     \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7960   ) ;
7961 }
7962 \end { tikzpicture }
7963 }
7964 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7965 \keys_define:nn { NiceMatrix / SpecialOffset }
7966 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7967 \cs_set_protected:Npn \@@_renew_matrix:
7968 {
7969   \RenewDocumentEnvironment { pmatrix } { } {
7970     { \pNiceMatrix }
7971     { \endpNiceMatrix }
7972   \RenewDocumentEnvironment { vmatrix } { } {
7973     { \vNiceMatrix }
7974     { \endvNiceMatrix }
7975   \RenewDocumentEnvironment { Vmatrix } { } {
7976     { \VNiceMatrix }
7977     { \endVNiceMatrix }
7978   \RenewDocumentEnvironment { bmatrix } { } {
7979     { \bNiceMatrix }
7980     { \endbNiceMatrix }
7981   \RenewDocumentEnvironment { Bmatrix } { } {
7982     { \BNiceMatrix }
7983     { \endBNiceMatrix }
7984 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7985 \keys_define:nn { NiceMatrix / Auto }
7986 {
7987   columns-type .tl_set:N = \l_@@_columns_type_tl ,
7988   columns-type .value_required:n = true ,
7989   l .meta:n = { columns-type = l } ,
7990   r .meta:n = { columns-type = r } ,
7991   c .meta:n = { columns-type = c } ,
7992   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

7993     delimiters / color .value_required:n = true ,
7994     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7995     delimiters / max-width .default:n = true ,
7996     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7997     delimiters .value_required:n = true ,
7998     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7999     rounded-corners .default:n = 4 pt
8000 }

8001 \NewDocumentCommand \AutoNiceMatrixWithDelims
8002 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8003 { \@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8004 \cs_new_protected:Npn \@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8005 {

```

The group is for the protection of the keys.

```

8006     \group_begin:
8007     \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
8008     \use:e
8009     {
8010         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8011         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8012         [ \exp_not:o \l_tmpa_tl ]
8013     }
8014     \int_if_zero:nT \l_@@_first_row_int
8015     {
8016         \int_if_zero:nT \l_@@_first_col_int { & }
8017         \prg_replicate:nn { #4 - 1 } { & }
8018         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8019     }
8020     \prg_replicate:nn { #3 }
8021     {
8022         \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).

```

8023         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8024         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8025     }
8026     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8027     {
8028         \int_if_zero:nT \l_@@_first_col_int { & }
8029         \prg_replicate:nn { #4 - 1 } { & }
8030         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8031     }
8032     \end { NiceArrayWithDelims }
8033     \group_end:
8034 }

8035 \cs_set_protected:Npn \@_define_com:nnn #1 #2 #3
8036 {
8037     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8038     {
8039         \bool_gset_true:N \g_@@_delims_bool
8040         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8041         \AutoNiceMatrixWithDelims { #2 } { #3 }
8042     }
8043 }

8044 \@_define_com:nnn p ( )
8045 \@_define_com:nnn b [ ]
8046 \@_define_com:nnn v | |
8047 \@_define_com:nnn V \ | \ |
8048 \@_define_com:nnn B \{ \}

```


We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8049 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8050 {
8051   \group_begin:
8052   \bool_gset_false:N \g_@@_delims_bool
8053   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8054   \group_end:
8055 }

```

30 The redefinition of the command `\dotfill`

```

8056 \cs_set_eq:NN \@@_old_dotfill \dotfill
8057 \cs_new_protected:Npn \@@_dotfill:
8058 {

```

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}`).

```

8059   \@@_old_dotfill
8060   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8061 }

```

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`.

```

8062 \cs_new_protected:Npn \@@_dotfill_i:
8063 { \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.

```

8064 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8065 {
8066   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8067   {
8068     \@@_actually_diagbox:nnnnnn
8069     { \int_use:N \c@iRow }
8070     { \int_use:N \c@jCol }
8071     { \int_use:N \c@iRow }
8072     { \int_use:N \c@jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

8073     { \g_@@_row_style_tl \exp_not:n { #1 } }
8074     { \g_@@_row_style_tl \exp_not:n { #2 } }
8075   }

```

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`.

```

8076   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8077   {
8078     { \int_use:N \c@iRow }
8079     { \int_use:N \c@jCol }
8080     { \int_use:N \c@iRow }
8081     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8082     { }
8083   }
8084 }

```

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.

```

8085 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8086 {
8087   \pgfpicture
8088   \pgf@relevantforpicturesizefalse
8089   \pgfrememberpicturepositiononpagetrue
8090   \@@_qpoint:n { row - #1 }
8091   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8092   \@@_qpoint:n { col - #2 }
8093   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8094   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8095   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8096   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8097   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8098   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8099   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8100   {

```

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.

```

8101     \CT@arc@
8102     \pgfsetroundcap
8103     \pgfusepathqstroke
8104   }
8105   \pgfset { inner~sep = 1 pt }
8106   \pgfscope
8107   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8108   \pgfnode { rectangle } { south~west }
8109   {
8110     \begin { minipage } { 20 cm }
8111     \@@_math_toggle: #5 \@@_math_toggle:
8112     \end { minipage }
8113   }
8114   { }
8115   { }
8116 \endpgfscope
8117 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8118 \pgfnode { rectangle } { north~east }
8119 {
8120   \begin { minipage } { 20 cm }
8121   \raggedleft
8122   \@@_math_toggle: #6 \@@_math_toggle:
8123   \end { minipage }
8124 }
8125 { }
8126 { }
8127 \endpgfpicture
8128 }

```

32 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 82.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8129 \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 `\`.

```
8130 \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`.

```
8131 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8132 {
8133   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8134   \@@_CodeAfter_iv:n
8135 }
```

We catch the argument of the command `\end` (in `#1`).

```
8136 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8137 {
```

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.

```
8138   \str_if_eq:eeTF \@currenvir { #1 }
8139   { \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`.

```
8140   {
8141     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8142     \@@_CodeAfter_ii:n
8143   }
8144 }
```

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.

```
8145 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8146 {
8147   \pgfpicture
8148   \pgfrememberpicturepositiononpagetrue
8149   \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.

```

8150 \@@_qpoint:n { row - 1 }
8151 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8152 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8153 \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).

```

8154 \bool_if:nTF { #3 }
8155 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8156 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8157 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8158 {
8159   \cs_if_exist:cT
8160   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8161   {
8162     \pgfpointanchor
8163     { \@@_env: - ##1 - #2 }
8164     { \bool_if:nTF { #3 } { west } { east } }
8165     \dim_set:Nn \l_tmpa_dim
8166     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8167   }
8168 }

```

Now we can put the delimiter with a node of PGF.

```

8169 \pgfset { inner~sep = \c_zero_dim }
8170 \dim_zero:N \nulldelimiterspace
8171 \pgftransformshift
8172 {
8173   \pgfpoint
8174   { \l_tmpa_dim }
8175   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8176 }
8177 \pgfnode
8178 { rectangle }
8179 { \bool_if:nTF { #3 } { east } { west } }
8180 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8181 \nullfont
8182 \c_math_toggle_token
8183 \@@_color:o \l_@@_delimiters_color_tl
8184 \bool_if:nTF { #3 } { \left #1 } { \left . }
8185 \vcenter
8186 {
8187   \nullfont
8188   \hrule \@height
8189   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8190   \@depth \c_zero_dim
8191   \@width \c_zero_dim
8192 }
8193 \bool_if:nTF { #3 } { \right . } { \right #1 }
8194 \c_math_toggle_token
8195 }
8196 { }
8197 { }
8198 \endpgfpicture
8199 }

```

34 The command \SubMatrix

```

8200 \keys_define:nn { NiceMatrix / sub-matrix }
8201 {
8202   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8203   extra-height .value_required:n = true ,
8204   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8205   left-xshift .value_required:n = true ,
8206   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8207   right-xshift .value_required:n = true ,
8208   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8209   xshift .value_required:n = true ,
8210   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8211   delimiters / color .value_required:n = true ,
8212   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8213   slim .default:n = true ,
8214   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8215   hlines .default:n = all ,
8216   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8217   vlines .default:n = all ,
8218   hvlines .meta:n = { hlines, vlines } ,
8219   hvlines .value_forbidden:n = true
8220 }
8221 \keys_define:nn { NiceMatrix }
8222 {
8223   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8224   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8225   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8226   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8227 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8228 \keys_define:nn { NiceMatrix / SubMatrix }
8229 {
8230   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8231   delimiters / color .value_required:n = true ,
8232   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8233   hlines .default:n = all ,
8234   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8235   vlines .default:n = all ,
8236   hvlines .meta:n = { hlines, vlines } ,
8237   hvlines .value_forbidden:n = true ,
8238   name .code:n =
8239     \tl_if_empty:nTF { #1 }
8240     { \@@_error:n { Invalid-name } }
8241     {
8242       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8243       {
8244         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8245         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8246         {
8247           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8248           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8249         }
8250       }
8251       { \@@_error:n { Invalid-name } }
8252     } ,
8253   name .value_required:n = true ,
8254   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8255   rules .value_required:n = true ,
8256   code .tl_set:N = \l_@@_code_tl ,

```

```

8257     code .value_required:n = true ,
8258     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8259 }

8260 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8261 {
8262     \peek_remove_spaces:n
8263     {
8264         \tl_gput_right:Nx \g_@@_pre_code_after_tl
8265         {
8266             \SubMatrix { #1 } { #2 } { #3 } { #4 }
8267             [
8268                 delimiters / color = \l_@@_delimiters_color_tl ,
8269                 hlines = \l_@@_submatrix_hlines_clist ,
8270                 vlines = \l_@@_submatrix_vlines_clist ,
8271                 extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8272                 left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8273                 right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8274                 slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8275                 #5
8276             ]
8277         }
8278         \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8279     }
8280 }

8281 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8282 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8283 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8284 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8285 {
8286     \seq_gput_right:Nx \g_@@_submatrix_seq
8287     {
We use \str_if_eq:nnTF because it is fully expandable.
8288         { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8289         { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8290         { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8291         { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8292     }
8293 }

```

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.

```

8294 \hook_gput_code:nnn { begindocument } { . }
8295 {
8296     \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8297     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

8298 \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8299 {
8300   \peek_remove_spaces:n
8301   {
8302     \@@_sub_matrix:nnnnnnn
8303     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8304   }
8305 }
8306 }

```

The following macro will compute $\backslash l_@@_first_i_tl$, $\backslash l_@@_first_j_tl$, $\backslash l_@@_last_i_tl$ and $\backslash l_@@_last_j_tl$ from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

8307 \NewDocumentCommand \@@_compute_i_j:nn
8308 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8309 { \@@_compute_i_j:nnnn #1 #2 }

8310 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8311 {
8312   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8313   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8314   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8315   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8316   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8317   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8318   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8319   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8320   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8321   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8322   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8323   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8324 }

8325 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8326 {
8327   \group_begin:

```

The four following token lists correspond to the position of the \backslash SubMatrix.

```

8328 \@@_compute_i_j:nn { #2 } { #3 }
8329 \int_compare:NnT \l_@@_first_i_tl = \l_@@_last_i_tl
8330 { \cs_set_nopar:Npn \arraystretch { 1 } }
8331 \bool_lazy_or:nnTF
8332 { \int_compare_p:Nn \l_@@_last_i_tl > \g_@@_row_total_int }
8333 { \int_compare_p:Nn \l_@@_last_j_tl > \g_@@_col_total_int }
8334 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8335 {
8336   \str_clear_new:N \l_@@_submatrix_name_str
8337   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8338   \pgfpicture
8339   \pgfrememberpicturepositiononpagetrue
8340   \pgf@relevantforpicturesizefalse
8341   \pgfset { inner~sep = \c_zero_dim }
8342   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8343   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \backslash int_step_inline:nnn is provided by curriification.

```

8344 \bool_if:NTF \l_@@_submatrix_slim_bool
8345 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8346 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8347 {
8348   \cs_if_exist:cT
8349   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8350   {
8351     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8352     \dim_set:Nn \l_@@_x_initial_dim

```

```

8353         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8354     }
8355     \cs_if_exist:cT
8356     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8357     {
8358         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8359         \dim_set:Nn \l_@@_x_final_dim
8360         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8361     }
8362 }
8363 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8364 { \@@_error:nn { Impossible~delimiter } { left } }
8365 {
8366     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8367     { \@@_error:nn { Impossible~delimiter } { right } }
8368     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8369 }
8370 \endpgfpicture
8371 }
8372 \group_end:
8373 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8374 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8375 {
8376     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8377     \dim_set:Nn \l_@@_y_initial_dim
8378     {
8379         \fp_to_dim:n
8380         {
8381             \pgf@y
8382             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8383         }
8384     }
8385     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8386     \dim_set:Nn \l_@@_y_final_dim
8387     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8388     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8389     {
8390         \cs_if_exist:cT
8391         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8392         {
8393             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8394             \dim_set:Nn \l_@@_y_initial_dim
8395             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8396         }
8397         \cs_if_exist:cT
8398         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8399         {
8400             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8401             \dim_set:Nn \l_@@_y_final_dim
8402             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8403         }
8404     }
8405     \dim_set:Nn \l_tmpa_dim
8406     {
8407         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8408         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8409     }
8410     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.


```

8411 \group_begin:
8412 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8413 \@@_set_CT@arc@:o \l_@@_rules_color_tl
8414 \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`.

```

8415 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8416 {
8417   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8418   {
8419     \int_compare:nNnT
8420       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8421     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8422       \@@_qpoint:n { col - ##1 }
8423       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8424       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8425       \pgfusepathqstroke
8426     }
8427   }
8428 }

```

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.

```

8429 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8430 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8431 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8432 {
8433   \bool_lazy_and:nnTF
8434     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8435     {
8436       \int_compare_p:nNn
8437         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8438       {
8439         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8440         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8441         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8442         \pgfusepathqstroke
8443       }
8444       { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8445     }

```

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.

```

8446 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8447 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8448 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8449 {
8450   \bool_lazy_and:nnTF
8451     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8452     {
8453       \int_compare_p:nNn
8454         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8455       {
8456         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8457 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8458 \dim_set:Nn \l_tmpa_dim

```

```

8459         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8460     \str_case:nn { #1 }
8461     {
8462         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8463         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8464         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8465         }
8466     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8467     \dim_set:Nn \l_tmpb_dim
8468     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8469     \str_case:nn { #2 }
8470     {
8471         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8472         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8473         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8474     }
8475     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8476     \pgfusepathqstroke
8477     \group_end:
8478 }
8479 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8480 }

```

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).

```

8481     \str_if_empty:NF \l_@@_submatrix_name_str
8482     {
8483         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8484         \l_@@_x_initial_dim \l_@@_y_initial_dim
8485         \l_@@_x_final_dim \l_@@_y_final_dim
8486     }
8487     \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`.

```

8488     \begin { pgfscope }
8489     \pgftransformshift
8490     {
8491         \pgfpoint
8492         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8493         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8494     }
8495     \str_if_empty:NTF \l_@@_submatrix_name_str
8496     { \@@_node_left:nn #1 { } }
8497     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8498     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8499     \pgftransformshift
8500     {
8501         \pgfpoint
8502         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8503         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8504     }
8505     \str_if_empty:NTF \l_@@_submatrix_name_str
8506     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8507     {
8508         \@@_node_right:nnnn #2
8509         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8510     }

```

```

8511 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8512 \flag_clear_new:n { nicematrix }
8513 \l_@@_code_tl
8514 }

```

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`.

```

8515 \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.

```

8516 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8517 {
8518   \use:e
8519   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8520 }

```

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 -.

```

8521 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8522 { #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.

```

8523 \tl_const:Nn \c_@@_integers_alist_tl
8524 {
8525   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8526   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8527   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8528   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8529 }

```

```

8530 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8531 {

```

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`.

```

8532 \tl_if_empty:nTF { #2 }
8533 {
8534   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8535   {
8536     \flag_raise:n { nicematrix }
8537     \int_if_even:nTF { \flag_height:n { nicematrix } }
8538     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8539     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8540   }
8541   { #1 }
8542 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```
8543     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8544 }
```

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`).

```
8545 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8546 {
8547   \str_case:nnF { #1 }
8548   {
8549     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8550     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8551   }
```

Now the case of a node of the form $i-j$.

```
8552   {
8553     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8554     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8555   }
8556 }
```

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`).

```
8557 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8558 {
8559   \pgfnode
8560   { rectangle }
8561   { east }
8562   {
8563     \nullfont
8564     \c_math_toggle_token
8565     \@@_color:o \l_@@_delimiters_color_tl
8566     \left #1
8567     \vcenter
8568     {
8569       \nullfont
8570       \hrule \@height \l_tmpa_dim
8571       \@depth \c_zero_dim
8572       \@width \c_zero_dim
8573     }
8574     \right .
8575     \c_math_toggle_token
8576   }
8577   { #2 }
8578   { }
8579 }
```

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.

```
8580 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8581 {
8582   \pgfnode
8583   { rectangle }
8584   { west }
8585   {
8586     \nullfont
8587     \c_math_toggle_token
8588     \@@_color:o \l_@@_delimiters_color_tl
8589     \left .
8590     \vcenter
```

```

8591      {
8592      \nullfont
8593      \hrule \@height \l_tmpa_dim
8594      \@depth \c_zero_dim
8595      \@width \c_zero_dim
8596      }
8597      \right #1
8598      \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8599      ^ { \smash { #4 } }
8600      \c_math_toggle_token
8601    }
8602    { #2 }
8603    { }
8604  }

```

35 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

8605 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8606 {
8607   \peek_remove_spaces:n
8608   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8609 }
8610 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8611 {
8612   \peek_remove_spaces:n
8613   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8614 }
8615 \keys_define:nn { NiceMatrix / Brace }
8616 {
8617   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8618   left-shorten .default:n = true ,
8619   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8620   shorten .meta:n = { left-shorten , right-shorten } ,
8621   right-shorten .default:n = true ,
8622   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8623   yshift .value_required:n = true ,
8624   yshift .initial:n = \c_zero_dim ,
8625   color .tl_set:N = \l_tmpa_tl ,
8626   color .value_required:n = true ,
8627   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8628 }

```

#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*.

```

8629 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8630 {
8631   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8632 \@@_compute_i_j:nn { #1 } { #2 }
8633 \bool_lazy_or:nnTF
8634 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8635 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8636 {
8637   \str_if_eq:nnTF { #5 } { under }

```

```

8638 { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8639 { \@@_error:nn { Construct-too-large } { \OverBrace } }
8640 }
8641 {
8642   \tl_clear:N \l_tmpa_tl
8643   \keys_set:nn { NiceMatrix / Brace } { #4 }
8644   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8645   \pgfpicture
8646   \pgfrememberpicturepositiononpagetrue
8647   \pgf@relevantforpicturesizefalse
8648   \bool_if:NT \l_@@_brace_left_shorten_bool
8649   {
8650     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8651     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8652     {
8653       \cs_if_exist:cT
8654       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8655       {
8656         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8657         \dim_set:Nn \l_@@_x_initial_dim
8658         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8659       }
8660     }
8661   }
8662   \bool_lazy_or:nnT
8663   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8664   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8665   {
8666     \@@_qpoint:n { col - \l_@@_first_j_tl }
8667     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8668   }
8669   \bool_if:NT \l_@@_brace_right_shorten_bool
8670   {
8671     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8672     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8673     {
8674       \cs_if_exist:cT
8675       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8676       {
8677         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8678         \dim_set:Nn \l_@@_x_final_dim
8679         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8680       }
8681     }
8682   }
8683   \bool_lazy_or:nnT
8684   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8685   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8686   {
8687     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8688     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8689   }
8690   \pgfset { inner~sep = \c_zero_dim }
8691   \str_if_eq:nnTF { #5 } { under }
8692   { \@@_underbrace_i:n { #3 } }
8693   { \@@_overbrace_i:n { #3 } }
8694   \endpgfpicture
8695 }
8696 \group_end:
8697 }

```

The argument is the text to put above the brace.

```

8698 \cs_new_protected:Npn \@@_overbrace_i:n #1
8699 {

```

```

8700 \@@_qpoint:n { row - \l_@@_first_i_tl }
8701 \pgftransformshift
8702 {
8703   \pgfpoint
8704   { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8705   { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
8706 }
8707 \pgfnode
8708 { rectangle }
8709 { south }
8710 {
8711   \vtop
8712   {
8713     \group_begin:
8714     \everycr { }
8715     \halign
8716     {
8717       \hfil ## \hfil \crcr
8718       \@@_math_toggle: #1 \@@_math_toggle: \cr
8719       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8720       \c_math_toggle_token
8721       \overbrace
8722       {
8723         \hbox_to_wd:nn
8724         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8725         { }
8726       }
8727       \c_math_toggle_token
8728       \cr
8729     }
8730     \group_end:
8731   }
8732 }
8733 { }
8734 { }
8735 }

```

The argument is the text to put under the brace.

```

8736 \cs_new_protected:Npn \@@_underbrace_i:n #1
8737 {
8738   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8739   \pgftransformshift
8740   {
8741     \pgfpoint
8742     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8743     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8744   }
8745   \pgfnode
8746   { rectangle }
8747   { north }
8748   {
8749     \group_begin:
8750     \everycr { }
8751     \vbox
8752     {
8753       \halign
8754       {
8755         \hfil ## \hfil \crcr
8756         \c_math_toggle_token
8757         \underbrace
8758         {
8759           \hbox_to_wd:nn
8760           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8761           { }

```

```

8762         }
8763         \c_math_toggle_token
8764         \cr
8765         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8766         \@@_math_toggle: #1 \@@_math_toggle: \cr
8767     }
8768 }
8769 \group_end:
8770 }
8771 { }
8772 { }
8773 }

```

36 The command TikzEveryCell

```

8774 \bool_new:N \l_@@_not_empty_bool
8775 \bool_new:N \l_@@_empty_bool
8776
8777 \keys_define:nn { NiceMatrix / TikzEveryCell }
8778 {
8779     not-empty .code:n =
8780         \bool_lazy_or:nnTF
8781             \l_@@_in_code_after_bool
8782             \g_@@_recreate_cell_nodes_bool
8783             { \bool_set_true:N \l_@@_not_empty_bool }
8784             { \@@_error:n { detection-of-empty-cells } } ,
8785     not-empty .value_forbidden:n = true ,
8786     empty .code:n =
8787         \bool_lazy_or:nnTF
8788             \l_@@_in_code_after_bool
8789             \g_@@_recreate_cell_nodes_bool
8790             { \bool_set_true:N \l_@@_empty_bool }
8791             { \@@_error:n { detection-of-empty-cells } } ,
8792     empty .value_forbidden:n = true ,
8793     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
8794 }
8795
8796
8797 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8798 {
8799     \IfPackageLoadedTF { tikz }
8800     {
8801         \group_begin:
8802         \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.

```

8803         \tl_set:Nn \l_tmpa_tl { { #2 } }
8804         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8805             { \@@_for_a_block:nnnnn #1 }
8806         \@@_all_the_cells:
8807         \group_end:
8808     }
8809     { \@@_error:n { TikzEveryCell-without-tikz } }
8810 }
8811
8812 \tl_new:N \@@_i_tl
8813 \tl_new:N \@@_j_tl
8814
8815 \cs_new_protected:Nn \@@_all_the_cells:

```



```

8816 {
8817   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8818   {
8819     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8820     {
8821       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8822       {
8823         \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8824         { \@@_i_tl - \@@_j_tl }
8825         {
8826           \bool_set_false:N \l_tmpa_bool
8827           \cs_if_exist:cTF
8828           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8829           {
8830             \bool_if:NF \l_@@_empty_bool
8831             { \bool_set_true:N \l_tmpa_bool }
8832           }
8833           {
8834             \bool_if:NF \l_@@_not_empty_bool
8835             { \bool_set_true:N \l_tmpa_bool }
8836           }
8837           \bool_if:NT \l_tmpa_bool
8838           {
8839             \@@_block_tikz:nnnnV
8840             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8841           }
8842         }
8843       }
8844     }
8845   }
8846 }
8847
8848 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8849 {
8850   \bool_if:NF \l_@@_empty_bool
8851   {
8852     \@@_block_tikz:nnnnV
8853     { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8854   }
8855   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8856 }
8857
8858 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8859 {
8860   \int_step_inline:nnn { #1 } { #3 }
8861   {
8862     \int_step_inline:nnn { #2 } { #4 }
8863     { \cs_set:cpn { cell - ##1 - ####1 } { } }
8864   }
8865 }

```

37 The command \ShowCellNames

```

8866 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8867 {
8868   \dim_zero_new:N \g_@@_tmpc_dim
8869   \dim_zero_new:N \g_@@_tmpd_dim
8870   \dim_zero_new:N \g_@@_tmpe_dim
8871   \int_step_inline:nn \c@iRow
8872   {
8873     \begin { pgfpicture }
8874     \@@_qpoint:n { row - ##1 }

```

```

8875 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8876 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8877 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8878 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8879 \bool_if:NTF \l_@@_in_code_after_bool
8880 \end { pgfpicture }
8881 \int_step_inline:nn \c@jCol
8882 {
8883   \hbox_set:Nn \l_tmpa_box
8884     { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
8885   \begin { pgfpicture }
8886     \@@_qpoint:n { col - ####1 }
8887     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8888     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8889     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8890     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8891     \endpgfpicture
8892   \end { pgfpicture }
8893   \fp_set:Nn \l_tmpa_fp
8894     {
8895       \fp_min:nn
8896       {
8897         \fp_min:nn
8898         {
8899           \dim_ratio:nn
8900             { \g_@@_tmpd_dim }
8901             { \box_wd:N \l_tmpa_box }
8902         }
8903         {
8904           \dim_ratio:nn
8905             { \g_tmpb_dim }
8906             { \box_ht_plus_dp:N \l_tmpa_box }
8907         }
8908       }
8909       { 1.0 }
8910     }
8911   \box_scale:Nnn \l_tmpa_box
8912     { \fp_use:N \l_tmpa_fp }
8913     { \fp_use:N \l_tmpa_fp }
8914   \pgfpicture
8915   \pgfrememberpicturepositiononpagetrue
8916   \pgf@relevantforpicturesizefalse
8917   \pgftransformshift
8918     {
8919       \pgfpoint
8920         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8921         { \dim_use:N \g_tmpa_dim }
8922     }
8923   \pgfnode
8924     { rectangle }
8925     { center }
8926     { \box_use:N \l_tmpa_box }
8927     { }
8928     { }
8929   \endpgfpicture
8930 }
8931 }
8932 }
8933 \NewDocumentCommand \@@_ShowCellNames { }
8934 {
8935   \bool_if:NT \l_@@_in_code_after_bool
8936   {
8937     \pgfpicture

```

```

8938 \pgfrememberpicturepositiononpagetrue
8939 \pgf@relevantforpicturesizefalse
8940 \pgfpathrectanglecorners
8941 { \@@_qpoint:n { 1 } }
8942 {
8943 \@@_qpoint:n
8944 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8945 }
8946 \pgfsetfillopacity { 0.75 }
8947 \pgfsetfillcolor { white }
8948 \pgfusepathqfill
8949 \endpgfpicture
8950 }
8951 \dim_zero_new:N \g_@@_tmpc_dim
8952 \dim_zero_new:N \g_@@_tmpd_dim
8953 \dim_zero_new:N \g_@@_tmpe_dim
8954 \int_step_inline:nn \c@iRow
8955 {
8956 \bool_if:NTF \l_@@_in_code_after_bool
8957 {
8958 \pgfpicture
8959 \pgfrememberpicturepositiononpagetrue
8960 \pgf@relevantforpicturesizefalse
8961 }
8962 { \begin { pgfpicture } }
8963 \@@_qpoint:n { row - ##1 }
8964 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8965 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8966 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8967 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8968 \bool_if:NTF \l_@@_in_code_after_bool
8969 { \endpgfpicture }
8970 { \end { pgfpicture } }
8971 \int_step_inline:nn \c@jCol
8972 {
8973 \hbox_set:Nn \l_tmpa_box
8974 {
8975 \normalfont \Large \sffamily \bfseries
8976 \bool_if:NTF \l_@@_in_code_after_bool
8977 { \color { red } }
8978 { \color { red ! 50 } }
8979 ##1 - ####1
8980 }
8981 \bool_if:NTF \l_@@_in_code_after_bool
8982 {
8983 \pgfpicture
8984 \pgfrememberpicturepositiononpagetrue
8985 \pgf@relevantforpicturesizefalse
8986 }
8987 { \begin { pgfpicture } }
8988 \@@_qpoint:n { col - ####1 }
8989 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8990 \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8991 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8992 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8993 \bool_if:NTF \l_@@_in_code_after_bool
8994 { \endpgfpicture }
8995 { \end { pgfpicture } }
8996 \fp_set:Nn \l_tmpa_fp
8997 {
8998 \fp_min:nn
8999 {
9000 \fp_min:nn

```

```

9001         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9002         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9003     }
9004     { 1.0 }
9005 }
9006 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9007 \pgfpicture
9008 \pgfrememberpicturepositiononpagetrue
9009 \pgf@relevantforpicturesizefalse
9010 \pgftransformshift
9011 {
9012     \pgfpoint
9013     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9014     { \dim_use:N \g_tmpa_dim }
9015 }
9016 \pgfnode
9017 { rectangle }
9018 { center }
9019 { \box_use:N \l_tmpa_box }
9020 { }
9021 { }
9022 \endpgfpicture
9023 }
9024 }
9025 }

```

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.

```

9026 \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.

```

9027 \bool_new:N \g_@@_footnote_bool

9028 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9029 {
9030     The~key~'\l_keys_key_str'~is~unknown. \\
9031     That~key~will~be~ignored. \\
9032     For~a~list~of~the~available~keys,~type~H~<return>.
9033 }
9034 {
9035     The~available~keys~are~(in~alphabetic~order):~
9036     footnote,~
9037     footnotehyper,~
9038     messages-for-Overleaf,~
9039     no-test-for-array,~
9040     renew-dots,~and~
9041     renew-matrix.
9042 }

9043 \keys_define:nn { NiceMatrix / Package }
9044 {
9045     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9046     renew-dots .value_forbidden:n = true ,
9047     renew-matrix .code:n = \@@_renew_matrix: ,

```

```

9048     renew-matrix .value_forbidden:n = true ,
9049     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9050     footnote .bool_set:N = \g_@@_footnote_bool ,
9051     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9052     no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
9053     no-test-for-array .default:n = true ,
9054     unknown .code:n = \@@_error:n { Unknown-key-for-package }
9055   }
9056 \ProcessKeysOptions { NiceMatrix / Package }

9057 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9058 {
9059   You-can't-use-the-option~'footnote'~because-the-package~
9060   footnotehyper-has-already-been-loaded.~
9061   If-you-want,-you-can-use-the-option~'footnotehyper'~and-the-footnotes~
9062   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9063   of-the-package-footnotehyper.\\
9064   The-package-footnote-won't-be-loaded.
9065 }

9066 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9067 {
9068   You-can't-use-the-option~'footnotehyper'~because-the-package~
9069   footnote-has-already-been-loaded.~
9070   If-you-want,-you-can-use-the-option~'footnote'~and-the-footnotes~
9071   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9072   of-the-package-footnote.\\
9073   The-package-footnotehyper-won't-be-loaded.
9074 }

9075 \bool_if:NT \g_@@_footnote_bool
9076 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9077   \IfClassLoadedTF { beamer }
9078   { \bool_set_false:N \g_@@_footnote_bool }
9079   {
9080     \IfPackageLoadedTF { footnotehyper }
9081     { \@@_error:n { footnote-with-footnotehyper-package } }
9082     { \usepackage { footnote } }
9083   }
9084 }

9085 \bool_if:NT \g_@@_footnotehyper_bool
9086 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9087   \IfClassLoadedTF { beamer }
9088   { \bool_set_false:N \g_@@_footnote_bool }
9089   {
9090     \IfPackageLoadedTF { footnote }
9091     { \@@_error:n { footnotehyper-with-footnote-package } }
9092     { \usepackage { footnotehyper } }
9093   }
9094   \bool_set_true:N \g_@@_footnote_bool
9095 }

```

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.

```
9096 \bool_new:N \l_@@_underscore_loaded_bool
9097 \IfPackageLoadedTF { underscore }
9098 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9099 { }
9100 \hook_gput_code:nnn { begindocument } { . }
9101 {
9102   \bool_if:NF \l_@@_underscore_loaded_bool
9103   {
9104     \IfPackageLoadedTF { underscore }
9105     { \@@_error:n { underscore~after~nicematrix } }
9106     { }
9107   }
9108 }
```

40 Error messages of the package

```
9109 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9110 { \str_const:Nn \c_@@_available_keys_str { } }
9111 {
9112   \str_const:Nn \c_@@_available_keys_str
9113   { For~a~list~of~the~available~keys,~type-H~<return>. }
9114 }
9115 \seq_new:N \g_@@_types_of_matrix_seq
9116 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9117 {
9118   NiceMatrix ,
9119   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9120 }
9121 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9122 { \tl_to_str:n { #1 } }
```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```
9123 \cs_new_protected:Npn \@@_error_too_much_cols:
9124 {
9125   \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9126   {
9127     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9128     { \@@_fatal:n { too~much~cols~for~matrix } }
9129     {
9130       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9131       { \@@_fatal:n { too~much~cols~for~matrix } }
9132       {
9133         \bool_if:NF \l_@@_last_col_without_value_bool
9134         { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9135       }
9136     }
9137   }
9138   { \@@_fatal:nn { too~much~cols~for~array } }
9139 }
```

The following command must *not* be protected since it's used in an error message.

```

9140 \cs_new:Npn \@@_message_hdotsfor:
9141 {
9142   \tl_if_empty:of \g_@@_HVDotsfor_lines_tl
9143   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9144 }
9145 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9146 {
9147   Incompatible~options.\\
9148   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9149   The~output~will~not~be~reliable.
9150 }
9151 \@@_msg_new:nn { negative~weight }
9152 {
9153   Negative~weight.\\
9154   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9155   the~value~'\int_use:N \l_@@_weight_int'.\\
9156   The~absolute~value~will~be~used.
9157 }
9158 \@@_msg_new:nn { last~col~not~used }
9159 {
9160   Column~not~used.\\
9161   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9162   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9163 }
9164 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9165 {
9166   Too~much~columns.\\
9167   In~the~row~\int_eval:n { \c@iRow },~
9168   you~try~to~use~more~columns~
9169   than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\
9170   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9171   (plus~the~exterior~columns).~This~error~is~fatal.
9172 }
9173 \@@_msg_new:nn { too~much~cols~for~matrix }
9174 {
9175   Too~much~columns.\\
9176   In~the~row~\int_eval:n { \c@iRow },~
9177   you~try~to~use~more~columns~than~allowed~by~your~
9178   \@@_full_name_env:.~\@@_message_hdotsfor:\ Recall~that~the~maximal~
9179   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9180   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9181   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9182   \token_to_str:N \setcounter\ to~change~that~value).~
9183   This~error~is~fatal.
9184 }
9185 \@@_msg_new:nn { too~much~cols~for~array }
9186 {
9187   Too~much~columns.\\
9188   In~the~row~\int_eval:n { \c@iRow },~
9189   ~you~try~to~use~more~columns~than~allowed~by~your~
9190   \@@_full_name_env:.~\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9191   \int_use:N \g_@@_static_num_of_col_int\
9192   ~(plus~the~potential~exterior~ones).
9193   This~error~is~fatal.
9194 }
9195 \@@_msg_new:nn { columns~not~used }
9196 {
9197   Columns~not~used.\\
9198   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N

```

```

9199 \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\
9200 The~columns~you~did~not~used~won't~be~created.\
9201 You~won't~have~similar~error~message~till~the~end~of~the~document.
9202 }
9203 \@@_msg_new:nn { in~first~col }
9204 {
9205   Erroneous~use.\
9206   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\
9207   That~command~will~be~ignored.
9208 }
9209 \@@_msg_new:nn { in~last~col }
9210 {
9211   Erroneous~use.\
9212   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\
9213   That~command~will~be~ignored.
9214 }
9215 \@@_msg_new:nn { in~first~row }
9216 {
9217   Erroneous~use.\
9218   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\
9219   That~command~will~be~ignored.
9220 }
9221 \@@_msg_new:nn { in~last~row }
9222 {
9223   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\
9224   That~command~will~be~ignored.
9225 }
9226 \@@_msg_new:nn { caption~outside~float }
9227 {
9228   Key~caption~forbidden.\
9229   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9230   environment.~This~key~will~be~ignored.
9231 }
9232 \@@_msg_new:nn { short~caption~without~caption }
9233 {
9234   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9235   However,~your~'short~caption'~will~be~used~as~'caption'.
9236 }
9237 \@@_msg_new:nn { double~closing~delimiter }
9238 {
9239   Double~delimiter.\
9240   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9241   delimiter.~This~delimiter~will~be~ignored.
9242 }
9243 \@@_msg_new:nn { delimiter~after~opening }
9244 {
9245   Double~delimiter.\
9246   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9247   delimiter.~That~delimiter~will~be~ignored.
9248 }
9249 \@@_msg_new:nn { bad~option~for~line~style }
9250 {
9251   Bad~line~style.\
9252   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9253   is~'standard'.~That~key~will~be~ignored.
9254 }
9255 \@@_msg_new:nn { Identical~notes~in~caption }
9256 {
9257   Identical~tabular~notes.\

```



```

9258     You~can't~put~several~notes~with~the~same~content~in~
9259     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\
9260     If~you~go~on,~the~output~will~probably~be~erroneous.
9261 }

9262 \@@_msg_new:nn { tabularnote~below~the~tabular }
9263 {
9264     \token_to_str:N \tabularnote\ forbidden\
9265     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9266     of~your~tabular~because~the~caption~will~be~composed~below~
9267     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9268     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\
9269     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9270     no~similar~error~will~raised~in~this~document.
9271 }

9272 \@@_msg_new:nn { Unknown~key~for~rules }
9273 {
9274     Unknown~key.\
9275     There~is~only~two~keys~available~here:~width~and~color.\
9276     Your~key~'\l_keys_key_str'~will~be~ignored.
9277 }

9278 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9279 {
9280     Unknown~key.\
9281     There~is~only~two~keys~available~here:~
9282     'empty'~and~'not~empty'~.\
9283     Your~key~'\l_keys_key_str'~will~be~ignored.
9284 }

9285 \@@_msg_new:nn { Unknown~key~for~rotate }
9286 {
9287     Unknown~key.\
9288     The~only~key~available~here~is~'c'~.\
9289     Your~key~'\l_keys_key_str'~will~be~ignored.
9290 }

9291 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9292 {
9293     Unknown~key.\
9294     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9295     It~you~go~on,~you~will~probably~have~other~errors. \
9296     \c_@@_available_keys_str
9297 }
9298 {
9299     The~available~keys~are~(in~alphabetic~order):~
9300     ccommand,~
9301     color,~
9302     command,~
9303     dotted,~
9304     letter,~
9305     multiplicity,~
9306     sep~color,~
9307     tikz,~and~total~width.
9308 }

9309 \@@_msg_new:nnn { Unknown~key~for~xdots }
9310 {
9311     Unknown~key.\
9312     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\
9313     \c_@@_available_keys_str
9314 }
9315 {
9316     The~available~keys~are~(in~alphabetic~order):~
9317     'color',~
9318     'horizontal~labels',~

```

```

9319     'inter',~
9320     'line-style',~
9321     'radius',~
9322     'shorten',~
9323     'shorten-end'~and~'shorten-start'.
9324 }

9325 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9326 {
9327     Unknown~key.\\
9328     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9329     (and~you~try~to~use~'\l_keys_key_str')\\
9330     That~key~will~be~ignored.
9331 }

9332 \@@_msg_new:nn { label~without~caption }
9333 {
9334     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9335     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9336 }

9337 \@@_msg_new:nn { W~warning }
9338 {
9339     Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9340     (row~\int_use:N \c@iRow).
9341 }

9342 \@@_msg_new:nn { Construct~too~large }
9343 {
9344     Construct~too~large.\\
9345     Your~command~\token_to_str:N #1
9346     can't~be~drawn~because~your~matrix~is~too~small.\\
9347     That~command~will~be~ignored.
9348 }

9349 \@@_msg_new:nn { underscore~after~nicematrix }
9350 {
9351     Problem~with~'underscore'.\\
9352     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9353     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9354     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9355 }

9356 \@@_msg_new:nn { ampersand~in~light-syntax }
9357 {
9358     Ampersand~forbidden.\\
9359     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9360     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9361 }

9362 \@@_msg_new:nn { double~backslash~in~light-syntax }
9363 {
9364     Double~backslash~forbidden.\\
9365     You~can't~use~\token_to_str:N
9366     \\~to~separate~rows~because~the~key~'light-syntax'~
9367     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9368     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9369 }

9370 \@@_msg_new:nn { hlines~with~color }
9371 {
9372     Incompatible~keys.\\
9373     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9374     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9375     May~be~it~will~be~possible~in~future~version.\\
9376     Your~key~will~be~discarded.
9377 }

9378 \@@_msg_new:nn { bad~value~for~baseline }

```

```

9379 {
9380     Bad-value-for-baseline.\\
9381     The-value-given-to-'baseline'~(\int_use:N \l_tmpa_int)~is-not~
9382     valid.~The-value-must-be-between-\int_use:N \l_@@_first_row_int\ and~
9383     \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'~or-of~
9384     the-form-'line-i'.\\
9385     A-value-of-1-will-be-used.
9386 }
9387 \@@_msg_new:nn { detection-of-empty-cells }
9388 {
9389     Problem-with-'not-empty'\\
9390     For-technical-reasons,~you-must-activate~
9391     'create-cell-nodes'~in-\token_to_str:N \CodeBefore\
9392     in-order-to-use-the-key~'\l_keys_key_str'.\\
9393     That-key-will-be-ignored.
9394 }
9395 \@@_msg_new:nn { siunitx-not-loaded }
9396 {
9397     siunitx-not-loaded\\
9398     You-can't-use-the-columns~'S'~because~'siunitx'~is-not-loaded.\\
9399     That-error-is-fatal.
9400 }
9401 \@@_msg_new:nn { ragged2e-not-loaded }
9402 {
9403     You-have-to-load~'ragged2e'~in-order-to-use-the-key~'\l_keys_key_str'~in~
9404     your-column~'\l_@@_vpos_col_str'~(or~'X').~The-key~'\str_lowercase:V
9405     \l_keys_key_str'~will-be-used-instead.
9406 }
9407 \@@_msg_new:nn { Invalid-name }
9408 {
9409     Invalid-name.\\
9410     You-can't-give-the-name~'\l_keys_value_tl'~to-a-\token_to_str:N
9411     \SubMatrix\ of-your~\@@_full_name_env:.\\
9412     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
9413     This-key-will-be-ignored.
9414 }
9415 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9416 {
9417     Wrong-line.\\
9418     You-try-to-draw-a-#1~line-of-number~'#2'~in-a~
9419     \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but-that~
9420     number-is-not-valid.~It-will-be-ignored.
9421 }
9422 \@@_msg_new:nn { Impossible-delimiter }
9423 {
9424     Impossible-delimiter.\\
9425     It's-impossible-to-draw-the-#1~delimiter-of-your~
9426     \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
9427     in-that-column.
9428     \bool_if:NT \l_@@_submatrix_slim_bool
9429     { ~Maybe-you-should-try-without-the-key~'slim'. } \\
9430     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9431 }
9432 \@@_msg_new:nnn { width-without-X-columns }
9433 {
9434     You-have-used-the-key~'width'~but-you-have-put-no~'X'~column.~
9435     That-key-will-be-ignored.
9436 }
9437 {
9438     This-message-is-the-message~'width-without-X-columns'~
9439     of-the-module~'nicematrix'.~

```

```

9440     The-experimented-users-can-disable-that-message-with-
9441     \token_to_str:N \msg_redirect_name:nnn.\\
9442   }
9443
9444   \@@_msg_new:nn { key-multiplicity-with-dotted }
9445   {
9446     Incompatible-keys. \\
9447     You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
9448     in~a~'custom-line'.~They-are-incompatible. \\
9449     The-key~'multiplicity'~will-be-discarded.
9450   }
9451
9452   \@@_msg_new:nn { empty-environment }
9453   {
9454     Empty-environment.\\
9455     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9456   }
9457
9458   \@@_msg_new:nn { No-letter-and-no-command }
9459   {
9460     Erroneous-use.\\
9461     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9462     key~'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys~'command'~or~
9463     ~'ccommand'~(to-draw-horizontal-rules).\\
9464     However,~you-can~go-on.
9465   }
9466
9467   \@@_msg_new:nn { Forbidden-letter }
9468   {
9469     Forbidden-letter.\\
9470     You-can't-use-the-letter~'#1'~for-a-customized-line.\\
9471     It-will-be-ignored.
9472   }
9473
9474   \@@_msg_new:nn { Several-letters }
9475   {
9476     Wrong-name.\\
9477     You-must-use-only-one-letter-as-value-for-the-key~'letter'~(and-you~
9478     have-used~'\l_@@_letter_str').\\
9479     It-will-be-ignored.
9480   }
9481
9482   \@@_msg_new:nn { Delimiter-with-small }
9483   {
9484     Delimiter-forbidden.\\
9485     You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\
9486     because-the-key~'small'~is-in-force.\\
9487     This-error-is-fatal.
9488   }
9489
9490   \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9491   {
9492     Unknown-cell.\\
9493     Your-command~\token_to_str:N\line\{#1\}\{#2\}~in~
9494     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9495     can't-be-executed-because-a-cell-doesn't-exist.\\
9496     This-command~\token_to_str:N \line\ will-be-ignored.
9497   }
9498
9499   \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9500   {
9501     Duplicate-name.\\
9502     The-name~'#1'~is-already-used-for-a~\token_to_str:N \SubMatrix\
9503     in~this~\@@_full_name_env:~\\
9504     This-key-will-be-ignored.\\
9505     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9506     { For-a-list-of-the-names-already-used,~type-H<return>. }
9507   }

```

```

9500 }
9501 {
9502   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9503   \seq~use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9504 }
9505 \@@_msg_new:nn { r~or~l~with~preamble }
9506 {
9507   Erroneous~use.\\
9508   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~.~
9509   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9510   your~\@@_full_name_env:~.\\
9511   This~key~will~be~ignored.
9512 }
9513 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9514 {
9515   Erroneous~use.\\
9516   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9517   the~array.~This~error~is~fatal.
9518 }
9519 \@@_msg_new:nn { bad~corner }
9520 {
9521   Bad~corner.\\
9522   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9523   'corners')~.~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9524   This~specification~of~corner~will~be~ignored.
9525 }
9526 \@@_msg_new:nn { bad~border }
9527 {
9528   Bad~border.\\
9529   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9530   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9531   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9532   also~use~the~key~'tikz'
9533   \IfPackageLoadedTF { tikz }
9534   { }
9535   {~if~you~load~the~LaTeX~package~'tikz'}).\\
9536   This~specification~of~border~will~be~ignored.
9537 }
9538 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9539 {
9540   TikZ~not~loaded.\\
9541   You~can't~use~\token_to_str:N \TikzEveryCell\
9542   because~you~have~not~loaded~tikz.~
9543   This~command~will~be~ignored.
9544 }
9545 \@@_msg_new:nn { tikz~key~without~tikz }
9546 {
9547   TikZ~not~loaded.\\
9548   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9549   \Block'~because~you~have~not~loaded~tikz.~
9550   This~key~will~be~ignored.
9551 }
9552 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9553 {
9554   Erroneous~use.\\
9555   In~the~\@@_full_name_env:,~you~must~use~the~key~
9556   'last~col'~without~value.\\
9557   However,~you~can~go~on~for~this~time~
9558   (the~value~'\l_keys_value_tl'~will~be~ignored).
9559 }

```

```

9560 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9561 {
9562   Erroneous-use.\
9563   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9564   'last-col'~without~value.\
9565   However,~you~can~go~on~for~this~time~
9566   (the~value~'\l_keys_value_tl'~will~be~ignored).
9567 }
9568 \@@_msg_new:nn { Block-too-large-1 }
9569 {
9570   Block-too-large.\
9571   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9572   too~small~for~that~block. \
9573   This~block~and~maybe~others~will~be~ignored.
9574 }
9575 \@@_msg_new:nn { Block-too-large-2 }
9576 {
9577   Block-too-large.\
9578   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9579   \g_@@_static_num_of_col_int\
9580   columns~but~you~use~only~\int_use:N \c_jCol\ and~that's~why~a~block~
9581   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9582   (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\
9583   This~block~and~maybe~others~will~be~ignored.
9584 }
9585 \@@_msg_new:nn { unknown-column-type }
9586 {
9587   Bad-column-type.\
9588   The~column~type~'#1'~in~your~\@@_full_name_env:\
9589   is~unknown. \
9590   This~error~is~fatal.
9591 }
9592 \@@_msg_new:nn { unknown-column-type-S }
9593 {
9594   Bad-column-type.\
9595   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \
9596   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9597   load~that~package. \
9598   This~error~is~fatal.
9599 }
9600 \@@_msg_new:nn { tabularnote-forbidden }
9601 {
9602   Forbidden-command.\
9603   You~can't~use~the~command~\token_to_str:N\ tabularnote\
9604   ~here.~This~command~is~available~only~in~
9605   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9606   the~argument~of~a~command~\token_to_str:N \caption\ included~
9607   in~an~environment~{table}. \
9608   This~command~will~be~ignored.
9609 }
9610 \@@_msg_new:nn { borders-forbidden }
9611 {
9612   Forbidden-key.\
9613   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9614   because~the~option~'rounded-corners'~
9615   is~in~force~with~a~non-zero~value.\
9616   This~key~will~be~ignored.
9617 }
9618 \@@_msg_new:nn { bottomrule-without-booktabs }
9619 {
9620   booktabs~not~loaded.\

```

```

9621     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9622     loaded~'booktabs'.\\
9623     This~key~will~be~ignored.
9624 }

9625 \@@_msg_new:nn { enumitem~not~loaded }
9626 {
9627     enumitem~not~loaded.\\
9628     You~can't~use~the~command~\token_to_str:N\tabularnote\
9629     ~because~you~haven't~loaded~'enumitem'.\\
9630     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9631     ignored~in~the~document.
9632 }

9633 \@@_msg_new:nn { tikz~without~tikz }
9634 {
9635     Tikz~not~loaded.\\
9636     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9637     loaded.~If~you~go~on,~that~key~will~be~ignored.
9638 }

9639 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9640 {
9641     Tikz~not~loaded.\\
9642     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9643     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9644     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9645     use~that~custom~line.
9646 }

9647 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9648 {
9649     Tikz~not~loaded.\\
9650     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9651     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9652     That~key~will~be~ignored.
9653 }

9654 \@@_msg_new:nn { without~color~inside }
9655 {
9656     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9657     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9658     outside~\token_to_str:N \CodeBefore,~you~
9659     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\\
9660     You~can~go~on~but~you~may~need~more~compilations.
9661 }

9662 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9663 {
9664     Erroneous~use.\\
9665     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9666     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9667     The~key~'color'~will~be~discarded.
9668 }

9669 \@@_msg_new:nn { Wrong~last~row }
9670 {
9671     Wrong~number.\\
9672     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9673     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9674     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9675     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9676     without~value~(more~compilations~might~be~necessary).
9677 }

9678 \@@_msg_new:nn { Yet~in~env }
9679 {
9680     Nested~environments.\\

```

```

9681     Environments-of~nicematrix~can't-be-nested.\\
9682     This~error~is~fatal.
9683 }
9684 \@@_msg_new:nn { Outside~math~mode }
9685 {
9686     Outside~math~mode.\\
9687     The~\@@_full_name_env:\ can-be-used-only~in~math~mode~
9688     (and~not~in~\token_to_str:N \vcenter).\\
9689     This~error~is~fatal.
9690 }
9691 \@@_msg_new:nn { One~letter~allowed }
9692 {
9693     Bad~name.\\
9694     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9695     It~will~be~ignored.
9696 }
9697 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9698 {
9699     Environment~{TabularNote}~forbidden.\\
9700     You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9701     but~*before*~the~\token_to_str:N \CodeAfter.\\
9702     This~environment~{TabularNote}~will~be~ignored.
9703 }
9704 \@@_msg_new:nn { varwidth~not~loaded }
9705 {
9706     varwidth~not~loaded.\\
9707     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9708     loaded.\\
9709     Your~column~will~behave~like~'p'.
9710 }
9711 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
9712 {
9713     Unknown~key.\\
9714     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9715     \c_@@_available_keys_str
9716 }
9717 {
9718     The~available~keys~are~(in~alphabetic~order):~
9719     color,~
9720     dotted,~
9721     multiplicity,~
9722     sep~color,~
9723     tikz,~and~total~width.
9724 }
9725
9726 \@@_msg_new:nnn { Unknown~key~for~Block }
9727 {
9728     Unknown~key.\\
9729     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9730     \Block.\\ It~will~be~ignored. \\
9731     \c_@@_available_keys_str
9732 }
9733 {
9734     The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9735     hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9736     respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9737 }
9738 \@@_msg_new:nnn { Unknown~key~for~Brace }
9739 {
9740     Unknown~key.\\
9741     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N

```



```

9742     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9743     It~will~be~ignored. \\
9744     \c_@@_available_keys_str
9745 }
9746 {
9747     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9748     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9749     right~shorten)~and~yshift.
9750 }
9751 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9752 {
9753     Unknown~key.\\
9754     The~key~'\l_keys_key_str'~is~unknown.\\
9755     It~will~be~ignored. \\
9756     \c_@@_available_keys_str
9757 }
9758 {
9759     The~available~keys~are~(in~alphabetic~order):~
9760     delimiters/color,~
9761     rules~(with~the~subkeys~'color'~and~'width'),~
9762     sub~matrix~(several~subkeys)~
9763     and~xdots~(several~subkeys).~
9764     The~latter~is~for~the~command~\token_to_str:N \line.
9765 }
9766 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9767 {
9768     Unknown~key.\\
9769     The~key~'\l_keys_key_str'~is~unknown.\\
9770     It~will~be~ignored. \\
9771     \c_@@_available_keys_str
9772 }
9773 {
9774     The~available~keys~are~(in~alphabetic~order):~
9775     create~cell~nodes,~
9776     delimiters/color~and~
9777     sub~matrix~(several~subkeys).
9778 }
9779 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9780 {
9781     Unknown~key.\\
9782     The~key~'\l_keys_key_str'~is~unknown.\\
9783     That~key~will~be~ignored. \\
9784     \c_@@_available_keys_str
9785 }
9786 {
9787     The~available~keys~are~(in~alphabetic~order):~
9788     'delimiters/color',~
9789     'extra~height',~
9790     'hlines',~
9791     'hvlines',~
9792     'left~xshift',~
9793     'name',~
9794     'right~xshift',~
9795     'rules'~(with~the~subkeys~'color'~and~'width'),~
9796     'slim',~
9797     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
9798     and~'right~xshift').\\
9799 }
9800 \@@_msg_new:nnn { Unknown~key~for~notes }
9801 {
9802     Unknown~key.\\
9803     The~key~'\l_keys_key_str'~is~unknown.\\

```

```

9804     That~key~will~be~ignored. \\
9805     \c_@@_available_keys_str
9806 }
9807 {
9808     The~available~keys~are~(in~alphabetic~order):~
9809     bottomrule,~
9810     code~after,~
9811     code~before,~
9812     detect~duplicates,~
9813     enumitem~keys,~
9814     enumitem~keys~para,~
9815     para,~
9816     label~in~list,~
9817     label~in~tabular~and~
9818     style.
9819 }

9820 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9821 {
9822     Unknown~key.\\
9823     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9824     \token_to_str:N \RowStyle. \\
9825     That~key~will~be~ignored. \\
9826     \c_@@_available_keys_str
9827 }
9828 {
9829     The~available~keys~are~(in~alphabetic~order):~
9830     'bold',~
9831     'cell~space~top~limit',~
9832     'cell~space~bottom~limit',~
9833     'cell~space~limits',~
9834     'color',~
9835     'nb~rows'~and~
9836     'rowcolor'.
9837 }

9838 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9839 {
9840     Unknown~key.\\
9841     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9842     \token_to_str:N \NiceMatrixOptions. \\
9843     That~key~will~be~ignored. \\
9844     \c_@@_available_keys_str
9845 }
9846 {
9847     The~available~keys~are~(in~alphabetic~order):~
9848     allow~duplicate~names,~
9849     caption~above,~
9850     cell~space~bottom~limit,~
9851     cell~space~limits,~
9852     cell~space~top~limit,~
9853     code~for~first~col,~
9854     code~for~first~row,~
9855     code~for~last~col,~
9856     code~for~last~row,~
9857     corners,~
9858     custom~key,~
9859     create~extra~nodes,~
9860     create~medium~nodes,~
9861     create~large~nodes,~
9862     delimiters~(several~subkeys),~
9863     end~of~row,~
9864     first~col,~
9865     first~row,~
9866     hlines,~

```

```

9867     hvlines,~
9868     hvlines-except-borders,~
9869     last-col,~
9870     last-row,~
9871     left-margin,~
9872     light-syntax,~
9873     light-syntax-expanded,~
9874     matrix/columns-type,~
9875     no-cell-nodes,~
9876     notes~(several~subkeys),~
9877     nullify-dots,~
9878     pgf-node-code,~
9879     renew-dots,~
9880     renew-matrix,~
9881     respect-arraystretch,~
9882     rounded-corners,~
9883     right-margin,~
9884     rules~(with~the~subkeys~'color'~and~'width'),~
9885     small,~
9886     sub-matrix~(several~subkeys),~
9887     vlines,~
9888     xdots~(several~subkeys).
9889 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9890 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9891 {
9892   Unknown~key.\\
9893   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9894   \{NiceArray\}. \\
9895   That~key~will~be~ignored. \\
9896   \c_@@_available_keys_str
9897 }
9898 {
9899   The~available~keys~are~(in~alphabetic~order):~
9900   b,~
9901   baseline,~
9902   c,~
9903   cell-space-bottom-limit,~
9904   cell-space-limits,~
9905   cell-space-top-limit,~
9906   code-after,~
9907   code-for-first-col,~
9908   code-for-first-row,~
9909   code-for-last-col,~
9910   code-for-last-row,~
9911   color-inside,~
9912   columns-width,~
9913   corners,~
9914   create-extra-nodes,~
9915   create-medium-nodes,~
9916   create-large-nodes,~
9917   extra-left-margin,~
9918   extra-right-margin,~
9919   first-col,~
9920   first-row,~
9921   hlines,~
9922   hvlines,~
9923   hvlines-except-borders,~
9924   last-col,~
9925   last-row,~
9926   left-margin,~
9927   light-syntax,~

```

```

9928     light-syntax-expanded,~
9929     name,~
9930     no-cell-nodes,~
9931     nullify-dots,~
9932     pgf-node-code,~
9933     renew-dots,~
9934     respect-arraystretch,~
9935     right-margin,~
9936     rounded-corners,~
9937     rules~(with~the~subkeys~'color'~and~'width'),~
9938     small,~
9939     t,~
9940     vl原因,~
9941     xdots/color,~
9942     xdots/shorten-start,~
9943     xdots/shorten-end,~
9944     xdots/shorten-and~
9945     xdots/line-style.
9946 }

```

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).

```

9947 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9948 {
9949   Unknown~key.\\
9950   The~key~'\l_keys_key_str'~is~unknown~for~the~
9951   \@@_full_name_env:. \\
9952   That~key~will~be~ignored. \\
9953   \c_@@_available_keys_str
9954 }
9955 {
9956   The~available~keys~are~(in~alphabetic~order):~
9957   b,~
9958   baseline,~
9959   c,~
9960   cell-space-bottom-limit,~
9961   cell-space-limits,~
9962   cell-space-top-limit,~
9963   code-after,~
9964   code-for-first-col,~
9965   code-for-first-row,~
9966   code-for-last-col,~
9967   code-for-last-row,~
9968   color-inside,~
9969   columns-type,~
9970   columns-width,~
9971   corners,~
9972   create-extra-nodes,~
9973   create-medium-nodes,~
9974   create-large-nodes,~
9975   extra-left-margin,~
9976   extra-right-margin,~
9977   first-col,~
9978   first-row,~
9979   hlines,~
9980   hvlines,~
9981   hvlines-except-borders,~
9982   l,~
9983   last-col,~
9984   last-row,~
9985   left-margin,~
9986   light-syntax,~
9987   light-syntax-expanded,~
9988   name,~

```

```

9989     no-cell-nodes,~
9990     nullify-dots,~
9991     pgf-node-code,~
9992     r,~
9993     renew-dots,~
9994     respect-arraystretch,~
9995     right-margin,~
9996     rounded-corners,~
9997     rules~(with~the~subkeys~'color'~and~'width'),~
9998     small,~
9999     t,~
10000    vlines,~
10001    xdots/color,~
10002    xdots/shorten-start,~
10003    xdots/shorten-end,~
10004    xdots/shorten-and~
10005    xdots/line-style.
10006  }
10007  \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10008  {
10009    Unknown~key.\\
10010    The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10011    \{NiceTabular\}. \\
10012    That~key~will~be~ignored. \\
10013    \c_@@_available_keys_str
10014  }
10015  {
10016    The~available~keys~are~(in~alphabetic~order):~
10017    b,~
10018    baseline,~
10019    c,~
10020    caption,~
10021    cell-space-bottom-limit,~
10022    cell-space-limits,~
10023    cell-space-top-limit,~
10024    code-after,~
10025    code-for-first-col,~
10026    code-for-first-row,~
10027    code-for-last-col,~
10028    code-for-last-row,~
10029    color-inside,~
10030    columns-width,~
10031    corners,~
10032    custom-line,~
10033    create-extra-nodes,~
10034    create-medium-nodes,~
10035    create-large-nodes,~
10036    extra-left-margin,~
10037    extra-right-margin,~
10038    first-col,~
10039    first-row,~
10040    hlines,~
10041    hvlines,~
10042    hvlines-except-borders,~
10043    label,~
10044    last-col,~
10045    last-row,~
10046    left-margin,~
10047    light-syntax,~
10048    light-syntax-expanded,~
10049    name,~
10050    no-cell-nodes,~
10051    notes~(several~subkeys),~

```

```

10052 nullify-dots,~
10053 pgf-node-code,~
10054 renew-dots,~
10055 respect-arraystretch,~
10056 right-margin,~
10057 rounded-corners,~
10058 rules~(with~the~subkeys~'color'~and~'width'),~
10059 short-caption,~
10060 t,~
10061 tabularnote,~
10062 vl原因,~
10063 xdots/color,~
10064 xdots/shorten-start,~
10065 xdots/shorten-end,~
10066 xdots/shorten-and~
10067 xdots/line-style.
10068 }

10069 \@@_msg_new:nnn { Duplicate-name }
10070 {
10071   Duplicate~name.\\
10072   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10073   the~same~environment~name~twice.~You~can~go~on,~but,~
10074   maybe,~you~will~have~incorrect~results~especially~
10075   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10076   message~again,~use~the~key~'allow-duplicate-names'~in~
10077   '\token_to_str:N \NiceMatrixOptions'.\\
10078   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10079     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10080 }
10081 {
10082   The~names~already~defined~in~this~document~are:~
10083   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10084 }

10085 \@@_msg_new:nn { Option~auto~for~columns~width }
10086 {
10087   Erroneous~use.\\
10088   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10089   That~key~will~be~ignored.
10090 }

10091 \@@_msg_new:nn { NiceTabularX~without~X }
10092 {
10093   NiceTabularX~without~X.\\
10094   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10095   However,~you~can~go~on.
10096 }

10097 \@@_msg_new:nn { Preamble~forgotten }
10098 {
10099   Preamble~forgotten.\\
10100   You~have~probably~forgotten~the~preamble~of~your~
10101   \@@_full_name_env:. \\
10102   This~error~is~fatal.
10103 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	3
3	Collecting options	4
4	Technical definitions	4
5	Parameters	9
6	The command <code>\tabularnote</code>	19
7	Command for creation of rectangle nodes	24
8	The options	25
9	Important code used by <code>{NiceArrayWithDelims}</code>	35
10	The <code>\CodeBefore</code>	49
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	58
13	The redefinition of <code>\multicolumn</code>	73
14	The environment <code>{NiceMatrix}</code> and its variants	90
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	91
16	After the construction of the array	93
17	We draw the dotted lines	99
18	The actual instructions for drawing the dotted lines with Tikz	112
19	User commands available in the new environments	118
20	The command <code>\line</code> accessible in code-after	124
21	The command <code>\RowStyle</code>	126
22	Colors of cells, rows and columns	128
23	The vertical and horizontal rules	140
24	The empty corners	155
25	The environment <code>{NiceMatrixBlock}</code>	158
26	The extra nodes	159
27	The blocks	163
28	How to draw the dotted lines transparently	183
29	Automatic arrays	183
30	The redefinition of the command <code>\dotfill</code>	185

31	The command <code>\diagbox</code>	185
32	The keyword <code>\CodeAfter</code>	187
33	The delimiters in the preamble	187
34	The command <code>\SubMatrix</code>	189
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	197
36	The command <code>TikzEveryCell</code>	200
37	The command <code>\ShowCellNames</code>	201
38	We process the options at package loading	204
39	About the package underscore	206
40	Error messages of the package	206